Entwurf und Implementierung eines Simulators für zeitdiskrete Simulation in C++

Ein Werkzeug zur Entwicklung wiederverwendbarer Modelle

Diplomarbeit

Band 2

Fachbereich Informatik

Universität Hamburg

vorgelegt von

Thomas Schniewind

im

März 1998

Erstbetreuer

Prof. Dr. Bernd Page

Zweitbetreuer

Prof. Dr. Rainer Lang

Programmtexte

accumula.h

```
// ----------------------------------------------------------------------------
//
// Datei
//          accumula.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ----------------------------------------------------------------------------

#ifndef ACCUMULATE_H
#define ACCUMULATE_H

// ----------------------------------------------------------------------------

#include "valuesta.h"   // Basisklasse

#include "str.h"

// ----------------------------------------------------------------------------

class Accumulate : public ValueStatistics
{
    public:
                    Accumulate (Model&          owner,
                        const   String&         name,
                                ValueSupplier&  vs,
                                bool            automatic    = false,
                                bool            showInReport = true,
                                bool            showInTrace  = false);
                    Accumulate (Model&          owner,
                                ValueSupplier&  vs,
                                bool            automatic    = false,
                                bool            showInReport = true,
                                bool            showInTrace  = false);

        virtual void            Update ();
        virtual void            Reset ();
        virtual double          Mean () const;
        virtual double          StdDev () const;
        virtual Reporter*       NewReporter () const;

                String          ClassName () const;

    private:
        double  sumTotal,
                sumSquareTotal;
        SimTime lastTime;
};

// ----------------------------------------------------------------------------

#endif  // ACCUMULATE_H
```

accumula.cc

```
// --------------------------------------------------------------------------
//
// Datei
//          accumula.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// --------------------------------------------------------------------------

#include "accumula.h"

#include "experimm.h"
#include <math.h>
#include "repstat.h"
#include "simclock.h"

// --------------------------------------------------------------------------

static const char* className = "Accumulate";

// --------------------------------------------------------------------------

Accumulate::Accumulate (Model&          owner,
                const   String&         name,
                        ValueSupplier&  vs,
                        bool            automatic,
                        bool            showInReport,
                        bool            showInTrace)
    :   ValueStatistics (owner, name, vs, showInReport, showInTrace),
        sumTotal        (0.0),
        sumSquareTotal  (0.0),
        lastTime        (0)
    {
        if (automatic)
            Observe (&ExperimentManager::Instance().GetSimClock(*this));
    }

// --------------------------------------------------------------------------

Accumulate::Accumulate (Model&          owner,
                        ValueSupplier&  vs,
                        bool            Automatic,
                        bool            showInReport,
                        bool            showInTrace)
    :   ValueStatistics (owner, "", vs, showInReport, showInTrace),
        sumTotal        (0.0),
        sumSquareTotal  (0.0),
        lastTime        (0)
    {
        if (Automatic)
            Observe (&ExperimentManager::Instance().GetSimClock(*this));
    }

// --------------------------------------------------------------------------

void Accumulate::Update ()
    {
        const char* where = "Accumulate::Update";

        if (!valid (className, where))
            return;

        SimTime now     = CurrentTime();
        SimTime diff    = now - lastTime;
        double  lastVal = Value();

        ValueStatistics::Update();

        sumTotal        += lastVal * diff.Time();
        sumSquareTotal  += lastVal * lastVal * diff.Time();
```

```
        lastTime        = now;
    }

// ------------------------------------------------------------------------------

void Accumulate::Reset ()
    {
        ValueStatistics::Reset();
        if (!Valid()) return;

        sumTotal        =
        sumSquareTotal  = 0;
        lastTime        = CurrentTime();
    }

// ------------------------------------------------------------------------------

double Accumulate::Mean () const
    {
        const char* where = "Accumulate::Mean";

        if (!valid (className, where))
            return -1.0;

        SimTime now         = CurrentTime();
        SimTime totalDiff   = now - ResetAt();

        if (totalDiff < Epsilon())
        {
            Warning (   "insufficient data",
                        where, "-1.0 is returned");
            return -1.0;
        }
        return  (sumTotal + Value() * (now - lastTime).Time())
                / totalDiff.Time();
    }

// ------------------------------------------------------------------------------

double Accumulate::StdDev () const
    {
        const char* where = "Accumulate::StdDev";

        if (!valid (className, where))
            return -1.0;

        SimTime now         = CurrentTime();
        SimTime diff        = now - lastTime;
        SimTime totalDiff   = now - ResetAt();

        if (totalDiff < Epsilon())
        {
            Warning (   "insufficient data",
                        where, "-1.0 is returned");
            return -1.0;
        }
        double  value   = Value();
        double  mean    = Mean();

        return  sqrt (fabs ((sumSquareTotal + value * value * diff.Time())
                / totalDiff.Time() - mean * mean));
    }

// ------------------------------------------------------------------------------

Reporter* Accumulate::NewReporter () const
    {
        return new AccumulateReporter (*this);
    }

// ------------------------------------------------------------------------------

String Accumulate::ClassName () const
    {
        return className;
    }

// ------------------------------------------------------------------------------
```

avl.h

```
// --------------------------------------------------------------------------
//
// Datei
//          avl.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// --------------------------------------------------------------------------
//
// Weiterentwicklung von:
//
// Diplomarbeit
//
//          Entwurf und Realisierung eines objektorientierten
//          Simulationspakets in C++
//
// Author
//          Heiko Weber
//
// Beschreibung
//
//          Ausgeglichene, binaere Baeume in C++ von N. Wirth
//
// --------------------------------------------------------------------------

#ifndef AVL_H
#define AVL_H

// --------------------------------------------------------------------------

/*
*
*   Die Struktur AvlNode wird in den Funktionen
*   avl_ins(), avl_insr(), avl_fnd(), avl_fndr(), avl_del(), avl_delr(),
*   avl_prt() und avl_prtr() intern verwendet. Der Anwender hat sich in
*   der Regel nicht darum zu kuemmern.
*   Die Information, die in einem ausgeglichenen binaeren Baum gespeichert
*   wird, ist ueber den char-Pointer 'daten' zu erreichen. Es handelt sich um
*   einen char-Pointer, um nicht auf eine Daten-Struktur festgelegt zu sein.
*   Die spezielle Daten-Struktur wird ueber eine mitzuliefernde Funktion
*   bearbeitet.
*/

enum   AvlBal  {
    LINKS = -1, MITTE = 0, RECHTS = 1
};

struct AvlNode {
    void    *daten;     /* Diese Komponente enthaelt den Verweis */
                        /* auf die Daten                         */
    AvlNode *left;
    AvlNode *right;
    AvlBal   balance;   /* Diese Komponente enthaelt die Werte   */
                        /* LINKS, MITTE, RECHTS                  */
                        /* Information fuer die Ausgeglichenheit */
                        /* des binaeren Baumes.                  */
             AvlNode(void *data)
             {
                 daten   = data;
                 left    = right = 0;
                 balance = MITTE;
             }
};

// --------------------------------------------------------------------------

enum AvlCmpResult {
                            LESS    = -1,
                            EQUAL   =  0,
                            GREATER =  1
```

```
};

// ------------------------------------------------------------------------

typedef AvlCmpResult      (*AvlCmp)(const void*, const void*);

// ------------------------------------------------------------------------

class Avl {
    AvlNode *tree;                  /* entry to tree                    */
    AvlNode *q;                     /* Dient der Kommunikation zwischen */
                                    /* delete(), balance1(), balance2() */
                                    /* und del().                       */
    AvlCmp   cmp;                   /* compare function                 */



    void                   *insert(void*, AvlNode**, int*);
    void                    links_ausgleichen(AvlNode**);
    void                    rechts_ausgleichen(AvlNode**);

    AvlNode                *search(void*, AvlNode*);
    void                   *remove(void*, AvlNode**, int*);
    void                    balance1(AvlNode**, int*);
    void                    balance2(AvlNode**, int*);
    void                    del(AvlNode**, int*);

    void                    print(int, void (*)(void*), AvlNode*);

    void                   *first(AvlNode*);
    void                   *next(void*, AvlNode*);
    void                   *last(AvlNode*);
    void                   *prev(void*, AvlNode*);

    void                    drop(AvlNode*);     // used to delete the tree

public:
                            Avl(AvlCmp cmpfunc)
                                : cmp(cmpfunc), tree(0), q(0) {}
    virtual                ~Avl(void) { drop(tree); }

    void                   *Insert(void *data)
                            {
                                int h;

                                return insert(data, &tree, &h);
                            }
    void                   *Search(void *data, AvlCmp cmp2 = 0)
                            {
                                AvlCmp old = cmp;
                                if (cmp2) cmp = cmp2;
                                AvlNode *root = search(data, tree);
                                cmp = old;
                                return (root) ? root->daten : 0;
                            }
    void                   *Remove(void *data)
                            {
                                int h;

                                return remove(data, &tree, &h);
                            }
    void                    Print(void (*prtfunc)(void*))
                            {
                                print(0, prtfunc, tree);
                            }
    void                   *First(void) { return first(tree); }
    void                   *Next(void *data, AvlCmp cmp2 = 0)
                            { AvlCmp old = cmp;
                              if (cmp2) cmp = cmp2;
                              void *ret = next(data, tree);
                              cmp = old;
                              return ret;
                            }
    void                   *Last(void) { return last(tree); }
    void                   *Prev(void *data) { return prev(data, tree); }
};

// ------------------------------------------------------------------------

#endif // AVL_H
```

avl.cc

```
// ---------------------------------------------------------------------------
//
// Datei
//          avl.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------
//
// Weiterentwicklung von:
//
// Diplomarbeit
//
//          Entwurf und Realisierung eines objektorientierten
//          Simulationspakets in C++
//
// Author
//          Heiko Weber
//
// Beschreibung
//
//          Ausgeglichene, binaere Baeume in C++ von N. Wirth
//
// ---------------------------------------------------------------------------

#include "avl.h"

#include "boolean.h"
#include <stdio.h>
#include <iostream.h>

// ---------------------------------------------------------------------------

void Avl::drop(AvlNode *root)
{
    if (!root) return;
    drop(root->left);
    drop(root->right);
    delete root;
}

// ---------------------------------------------------------------------------

void *Avl::insert(void *daten, AvlNode **root, int *h)
{
    void *s;

    if( *root == NULL ) {
        *root = new AvlNode(daten);
        *h = true ;
        return(daten);
    }
    else
    { switch( (*cmp)(daten,(*root)->daten) )
        {
            case    LESS:
                    s = insert(daten, &(*root)->left, h);
                    if( *h )    /* Der linke Ast wurde groesser         */
                    {
                        switch( (*root)->balance )
                        {
                            case    RECHTS: (*root)->balance = MITTE ;
                                            *h = false ;
                                            break ;

                            case    MITTE:  (*root)->balance = LINKS ;
                                            break ;

                            case    LINKS:  /* erneutes Ausgleichen     */
                                            links_ausgleichen(root) ;
```

```
                                        (*root)->balance = MITTE ;
                                        *h = false ;
                                        break ;
                        }       /* end of switch( (*root)->balance )    */
                    }           /* end of if(*h)                        */
                    break ;

            case    EQUAL:
                    *h = false ;
                    s = (*root)->daten ;
                    break ;

            case    GREATER:
                    s = insert(daten, &(*root)->right, h);
                    if( *h )    /* Der rechte Ast wurde groesser        */
                    {
                        switch( (*root)->balance )
                        {
                            case    RECHTS: /* erneutes Ausgleichen     */
                                        rechts_ausgleichen(root) ;
                                        (*root)->balance = MITTE ;
                                        *h = false ;
                                        break ;

                            case    MITTE:  (*root)->balance = RECHTS ;
                                        break ;

                            case    LINKS:  (*root)->balance = MITTE ;
                                        *h = false ;
                                        break ;
                        }       /* end of switch( (*root)->balance )    */
                    }           /* end of if(*h)                        */
                    break ;
        }           /* end of switch( (*cmp)(daten,(*root)->daten) )    */
        return(s) ;
    }
}

// ------------------------------------------------------------------------------

void Avl::links_ausgleichen(AvlNode **root)
{
    AvlNode *p1, *p2 ;

    p1 = (*root)->left ;

    if( p1->balance == LINKS )
    {                                           /* einfache LL Rotation */
        (*root)->left    = p1->right ;
         p1->right       = *root ;
        (*root)->balance = MITTE ;
         *root           = p1 ;
    }
    else
    {                                           /* doppelte LR Rotation */
         p2              = p1->right ;
         p1->right       = p2->left ;
         p2->left        = p1 ;
        (*root)->left    = p2->right ;
         p2->right       = *root ;
        (*root)->balance = ( p2->balance == LINKS  ) ? RECHTS : MITTE ;
         p1->balance     = ( p2->balance == RECHTS ) ? LINKS  : MITTE ;
         *root           = p2 ;
    }
}

// ------------------------------------------------------------------------------

void Avl::rechts_ausgleichen(AvlNode **root)
{
    AvlNode *p1, *p2 ;

    p1 = (*root)->right ;

    if( p1->balance == RECHTS )
    {                                           /* einfache RR Rotation */
        (*root)->right   = p1->left ;
         p1->left        = *root ;
        (*root)->balance = MITTE ;
         *root           = p1 ;
    }
    else
    {                                           /* doppelte RL Rotation */
         p2              = p1->left ;
```

```
         p1->left        = p2->right ;
         p2->right       = p1 ;
        (*root)->right   = p2->left ;
         p2->left        = *root ;
        (*root)->balance = ( p2->balance == RECHTS ) ? LINKS  : MITTE ;
         p1->balance     = ( p2->balance == LINKS  ) ? RECHTS : MITTE ;
         *root           = p2 ;
    }
}

// ------------------------------------------------------------------------------

AvlNode *Avl::search(void *daten, AvlNode *root)
{
    while (root) {
        switch((*cmp)(daten, root->daten)) {
            case    LESS:    root = root->left ;
                                break ;

            case    EQUAL:     return root;

            case    GREATER:   root = root->right ;
                                break ;

            default:        printf("parameter-error in avl_fndr(): 'cmp'\n");
                            fflush(stdout) ;
                            return 0;
        }
    }

    return 0;   /* Kein Eintrag vorhanden */
}

// ------------------------------------------------------------------------------

void *Avl::remove(void *daten, AvlNode **root, int *h)
{
    void *ret;

    if( *root == NULL )
    {                       /* Das Kriterium ist nicht im Baum eingetragen  */
        *h  = false;
        ret = 0;
    }
    else
    {
        switch( (*cmp)(daten,(*root)->daten) )
        {
            case   LESS: ret = remove(daten, &(*root)->left, h);
                            if( *h ) balance1(root,h) ;
                            break ;

            case    EQUAL: /* entferne den **root - Eintrag    */
                            q   = *root ;
                            ret = q->daten;
                            if( q->right == NULL )
                            {
                                *root = q->left ;
                                *h    = true ;
                            }
                            else
                            {
                                if( q->left == NULL )
                                {
                                    *root = q->right ;
                                    *h    = true ;
                                }
                                else
                                {
                                    del(&q->left, h);
                                    (*root)->daten = q->daten;
                                    if( *h ) balance1(root,h) ;
                                }
                            }
                            delete q;
                            break ;

            case  GREATER: ret = remove(daten, &(*root)->right, h);
                            if( *h ) balance2(root,h) ;
                            break ;

            default:        printf("parameter-error in avl_delr(): 'cmp'\n");
                            fflush(stdout) ;
                            break ;
```

```
        }   /* end of switch( (*cmp)() )     */
    }       /* end of if(*root == NULL)      */
    return ret;
}           /* end of remove()               */

// ----------------------------------------------------------------------------

void Avl::balance1(AvlNode **root, int *h)
{                               /* *h == true, der linke Ast wurde groesser */
    AvlNode *p1, *p2 ;
    int      b1,  b2 ;

    switch( (*root)->balance )
    {
        case    LINKS:
                (*root)->balance = MITTE ;
                break ;

        case    MITTE:
                (*root)->balance = RECHTS ;
                *h = false ;
                break ;

        case    RECHTS:                         /* erneutes Ausgleichen */
                p1 = (*root)->right ;
                b1 = p1->balance ;
                if( (b1 == MITTE) || (b1 == RECHTS) )
                {                               /* einfache RR Rotation */
                    (*root)->right = p1->left ;
                    p1->left = *root ;
                    if( b1 == MITTE )
                    {
                        (*root)->balance = RECHTS ;
                        p1->balance = LINKS ;
                        *h = false ;
                    }
                    else
                    {
                        (*root)->balance = MITTE ;
                        p1->balance = MITTE ;
                    }
                    *root = p1 ;
                }
                else
                {                               /* doppelte RL Rotation */
                    p2 = p1->left ;
                    b2 = p2->balance ;
                    p1->left = p2->right ;
                    p2->right = p1 ;
                    (*root)->right = p2->left ;
                    p2->left = *root ;
                    (*root)->balance = ( b2 == RECHTS ) ? LINKS : MITTE ;
                    p1->balance      = ( b2 == LINKS  ) ? RECHTS : MITTE ;
                    *root = p2 ;
                    p2->balance = MITTE ;
                }
                break ;
    }
}

// ----------------------------------------------------------------------------

void Avl::balance2(AvlNode **root, int *h)
{                              /* *h == true, der rechte Ast wurde groesser */
    AvlNode *p1, *p2 ;
    int      b1,  b2 ;

    switch( (*root)->balance )
    {
        case    LINKS:                          /* erneutes Ausgleichen */
                p1 = (*root)->left ;
                b1 = p1->balance ;
                if( (b1 == LINKS) || (b1 == MITTE) )
                {                               /* einfache LL Rotation */
                    (*root)->left = p1->right ;
                    p1->right = *root ;
                    if( b1 == MITTE )
                    {
                        (*root)->balance = LINKS ;
                        p1->balance = RECHTS ;
                        *h = false ;
                    }
                    else
                    {
```

```
                    (*root)->balance = MITTE ;
                    p1->balance = MITTE ;
                }
                *root = p1 ;
            }
            else
            {                                   /* doppelte LR Rotation */
                p2 = p1->right ;
                b2 = p2->balance ;
                p1->right = p2->left ;
                p2->left = p1 ;
                (*root)->left = p2->right ;
                p2->right = *root ;
                (*root)->balance = ( b2 == LINKS  ) ? RECHTS : MITTE ;
                p1->balance      = ( b2 == RECHTS ) ? LINKS  : MITTE ;
                *root = p2 ;
                p2->balance = MITTE ;
            }
            break ;

        case    MITTE:
                (*root)->balance = LINKS ;
                *h = false ;
                break ;

        case    RECHTS:
                (*root)->balance = MITTE ;
                break ;
    }
}

// ---------------------------------------------------------------------------

void Avl::del(AvlNode **r, int *h)
{
    if( (*r)->right != NULL )
    {
        del( &(*r)->right, h);
        if( *h ) balance2(r,h) ;
    } else {
        q = *r ;
        *r = (*r)->left ;
        *h = true ;
    }
}

// ---------------------------------------------------------------------------

/*
*   Diese Funktion gibt den ausgeglichenen binaeren Baum mit der Wurzel
*   'root' aus.
*   Die Funktion 'avl_prt()' kennt die eingetragene Datenstruktur nicht.
*   Aus diesem Grund wird an die Ausgabefunktion 'prtfunc()' ein
*   char-Pointer auf die eingetragenen Daten uebergeben. Die Augabefunktion
*   muss diesen char-Pointer mit dem cast-operator in den entsprechenden
*   struktur-Pointer umwandeln und die Daten, der Struktur ensprechend,
*   ausgeben.
*   level gibt die aktuelle Hoehe im Baum an. (Eingabewert: 1).
*   Fuer jeden Hoehewert > 1 werden 3 Blanks auf stdout ausgegeben, um die
*   Baumstruktur deutlich zu machen.
*
*   Rueckgabewert: Keiner
*
*/

void Avl::print(int level, void (*prtfunc)(void*), AvlNode *root)
{
    register int i ;

    if( root != NULL )
    {
        print(level+1,prtfunc,root->left) ;
        for(i=0; i<level; i++) cout << "   ";
        (*prtfunc)(root->daten) ;
        print(level+1,prtfunc,root->right) ;
    }
}

// ---------------------------------------------------------------------------

void *Avl::first(AvlNode *root)
{
    if (root) {
        while (root->left)
```

```
            root = root->left;
        return root->daten;
    }
    return 0;
}

// ------------------------------------------------------------------------------

void *Avl::last(AvlNode *root)
{
    if (root) {
        while (root->right)
            root = root->right;
        return root->daten;
    }
    return 0;
}

// ------------------------------------------------------------------------------

void *Avl::next(void *daten, AvlNode *root)
{
    if (root)
        switch((*cmp)(daten, root->daten)) {
            case LESS:
                {
                    void *data = next(daten, root->left);
                    return (data) ? data : root->daten;
                }
            case EQUAL:
                return first(root->right);
            case GREATER:
                return next(daten, root->right);
            default:
                printf("parameter-error in avl_fndr(): 'cmp'\n");
                fflush(stdout) ;
                return 0;
        }
    return 0;
}

// ------------------------------------------------------------------------------

void *Avl::prev(void *daten, AvlNode *root)
{
    if (root)
        switch((*cmp)(daten, root->daten)) {
            case LESS:
                return prev(daten, root->left);
            case EQUAL:
                return last(root->left);
            case GREATER:
                {
                    void *data = prev(daten, root->right);
                    return (data) ? data : root->daten;
                }
            default:
                printf("parameter-error in avl_fndr(): 'cmp'\n");
                fflush(stdout) ;
                return 0;
        }
    return 0;
}

// ------------------------------------------------------------------------------

#ifdef DEBUG

// ------------------------------------------------------------------------------

static int intcmp(const void *a, const void *b)
{
    int *aa = (int*) a;
    int *bb = (int*) b;

    if (*aa == *bb) return EQUAL;
    return (*aa < *bb) ? LESS : GREATER;
}

// ------------------------------------------------------------------------------

static void intprt(void *a)
{
    int *aa = (int*) a;
```

```
    printf("%d\n", *aa);
}

// ---------------------------------------------------------------------------

int test(void)
{
    Avl *a = new Avl(intcmp);
    int  *v, w;

    for (int i = 0; i < 3; i++) {
        v = new int(i);
        a->Insert((void*) v);
    }

    printf("Inserted : %4d\n", i);
    a->Print(intprt);
    i = 1;
    a->Remove(&i);
    printf("Delete: %4d\n", i);
    a->Print(intprt);
}

// ---------------------------------------------------------------------------

#include <alloc.h>

// ---------------------------------------------------------------------------

int main(void)
{
    Avl a(intcmp);

    printf("Coreleft = %lu\n", long(coreleft()));

    for (int i = 0; i < 1000; i++) {
        int *v = new int(i);
        a.Insert((void*) v);

        printf("Inserted : %4d\r", i);

        v = 0;
        for (void *w = a.First(); w; w = a.Next(w)) {
            if (v) {
                if (*v + 1 != *(int*)w) {
                    printf("Fehler bei Next: %d,%d\n", *v, *(int*)w);
                    getchar();
                }
            }
            v = (int*) w;
        }
        v = 0;
        for (w = a.Last(); w; w = a.Prev(w)) {
            if (v) {
                if (*v - 1 != *(int*)w) {
                    printf("Fehler bei Prev: %d,%d\n", *v, *(int*)w);
                    getchar();
                }
            }
            v = (int*) w;
        }
    }

    a.Print(intprt);

    printf("Coreleft = %lu after %d inserts\n", long(coreleft()), i);

    printf("press <return> : "); fflush(stdout);
    getchar();

    for (i = 0; i < 1000; i += 2) {
        int *v = (int*) a.Remove((void*) &i);
        if (v)
            delete v;
        else
            printf("oops: remove %d = 0\n", i);
        printf("%4d\r", i);
    }

    a.Print(intprt);

    printf("Coreleft = %lu after %d deletes\n", long(coreleft()), i / 2);
```

```
    printf("press <return> : "); fflush(stdout);
    getchar();

    for (i = 1; i < 1000; i += 2) {
        int *v = (int*) a.Remove((void*) &i);
        if (v)
            delete v;
        else
            printf("oops: remove %d = 0\n", i);
        printf("%4d\r", i);
    }

    a.Print(intprt);

    printf("Coreleft = %lu after %d deletes\n", long(coreleft()), i / 2);

    return 0;
}

// ---------------------------------------------------------------------------

#endif // DEBUG
```

bin.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          bin.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#ifndef BIN_H
#define BIN_H

// ---------------------------------------------------------------------------

#include "qbased.h" // Basisklasse

#include "simtime.h"
#include "str.h"

// ---------------------------------------------------------------------------

class QueueImpl;

// ---------------------------------------------------------------------------

class Bin : public QueueBased
{
        Bin&        operator= (const Bin&);
                        // Zuweisung nicht implementiert
    public:
                    Bin (       Model&          owner,
                        const   String&         name,
                                unsigned long   initial = 0,
                                bool            showInReport = true,
                                bool            showInTrace = true);
                    Bin (const Bin&);
        virtual    ~Bin ();

                void            Take (unsigned long n);
                void            Give (unsigned long n);

        virtual void            Reset();
```

```
                unsigned long   Producers () const;
                unsigned long   Consumers () const;
                unsigned long   Users () const      { return Producers(); };
                unsigned long   Initial () const;
                unsigned long   Maximum () const;
                unsigned long   Avail () const;
                double          AvgAvail () const;

        virtual Reporter*       NewReporter() const;

                String          ClassName () const;
    protected:

    private:
                void        updateStatistics (long n);
                                // betrifft nicht 'users'
                bool        checkProcess (      Process& p,
                                          const char*    where) const;
                void        activateNext (const char*    where) const;

        QueueImpl&      qimpl;
        unsigned long   initial,
                        maximum,
                        avail,
                        producers,
                        consumers;
        double          wSumAvail;
        SimTime         lastReturn;
};

// ---------------------------------------------------------------------------

#endif  // BIN_H
```

bin.cc

```
// ---------------------------------------------------------------------------
//
// Datei
//          bin.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#include "bin.h"

#include "pblocker.h"
#include "process.h"
#include "qimpl.h"
#include "represbi.h"
#include "msgresbi.h"

#include <assert.h>

// ---------------------------------------------------------------------------

static const char* className = "Bin";

// ---------------------------------------------------------------------------

Bin::Bin (      Model&          owner,
        const   String&         name,
                unsigned long   Initial,
                bool            showInReport,
                bool            showInTrace)
    :   QueueBased  (owner, name, showInReport, showInTrace),
        qimpl       (*new QueueImpl (*this)),
```

```
        initial     (Initial),
        maximum     (Initial),
        avail       (Initial),
        producers   (0),
        consumers   (0),
        wSumAvail   (0.0),
        lastReturn  (0.0)
    {}

// ---------------------------------------------------------------------------

// die implizite Warteschalnge ist leer (wird z.t. ueber QueueBased geregelt)
Bin::Bin (const Bin& b)
    :   QueueBased  (b),
        qimpl       (*new QueueImpl (*this)),
        initial     (b.initial),
        maximum     (b.initial),
        avail       (b.initial),
        producers   (0),
        consumers   (0),
        wSumAvail   (0.0),
        lastReturn  (0.0)
    {}

// ---------------------------------------------------------------------------

Bin::~Bin ()
    {
        delete &qimpl;
    }

// ---------------------------------------------------------------------------

void Bin::updateStatistics (long n)
    {
        // n positiv fuer 'Give()', negativ fuer 'Take()'
        SimTime now = CurrentTime();
        wSumAvail   += (double) avail * (now - lastReturn).Time();
        lastReturn   = now;
        avail       += n;

        if (n > 0)
        {
            producers++;
            if (avail > maximum)
                maximum = avail;
        }
        else
            consumers++;
    }

// ---------------------------------------------------------------------------

bool Bin::checkProcess (Process& p, const char* where) const
    {
        if (!p.Valid ())
        {
            Warning ("invalid object", where);
            return false;
        }
        if (p.IsNullProcess())
        {
            Warning ("only processes may give to or take from bins",
                     where);
            return false;
        }
        if (!IsExperimentCompatible (p))
        {
            Warning ("attemp to mix components of different experiments",
                     where, "ignored");
            return false;
        }
        if (!IsModelCompatible (p))
        {
            Warning ("incompatible process", where);
            return false;
        }
        return true;
    }

// ---------------------------------------------------------------------------

void Bin::activateNext (const char* where) const
    {
```

```
        if (Length() > 0)
        {
            Process& next = (Process&)qimpl.First (where);
            if (!checkProcess (next, where))
                return;
            if (next.IsScheduled())
            {   // anders als in DESMO!
                next.SkipTraceNote ();
                next.Cancel();
            }


            bool wasBlocked = next.Blocked();
            if (wasBlocked)
                ProcessBlocker::UnBlock (next);
                    // um Aktivierung zu erlauben

            next.SkipTraceNote ();
            next.ActivateAfter (Current());

            if (wasBlocked) ProcessBlocker::SetBlocked (next);
        }
    }

// ------------------------------------------------------------------------------

void Bin::Take (unsigned long n)
    {
        const char*     where = "Bin::Take";

        if (!valid (className, where))
            return;

        bool            activateSuccessor = false;
        Process&        process = CurrentProcess();
        if (!checkProcess (process, where))
            return;

        if (n <= 0)
        {
            if (TraceIsOn())
                SendMessage (TrcBinTake (*this, n));
            return;
        }

        qimpl.Insert (process, where);

        if (n > avail ||                     // nicht genug vorhanden oder
            process != qimpl.First (where)) // es wartet noch ein anderer
        {
            if (TraceIsOn())
                SendMessage (TrcResBinAwait (*this, n));

            do
            {
                ProcessBlocker::Block (process);
            } while (avail < n || process != qimpl.First (where));
                    // nicht genuegend frei oder ein anderer vorher dran

            activateSuccessor = true;
        }

        if (TraceIsOn())
            SendMessage (TrcBinTake (*this, n));

        //SkipTraceNote ();
        qimpl.Remove (process, where);  // aus impliziter Queue entfernen
        ProcessBlocker::UnBlock (process);

        // nachdem current aus Queue entfernt:
        if (activateSuccessor)
            activateNext (where);

        updateStatistics (-n);  // -n fuer 'Take (n)'
    }

// ------------------------------------------------------------------------------

void Bin::Give (unsigned long n)
    {
        const char* where = "Bin::Give";

        if (!valid (className, where))
            return;
```

```
        Process& process = CurrentProcess();
        if (!checkProcess (process, where))
            return;

        if (TraceIsOn())
            SendMessage (TrcBinGive (*this, n));

        updateStatistics (n);   // +n fuer 'Give (n)'

        activateNext (where);
    }

// ---------------------------------------------------------------------------

void Bin::Reset ()
    {
        const char* where = "Bin::Reset";

        QueueBased::Reset();
        if (!Valid ()) return;

        maximum     = avail;
        wSumAvail   =
        producers   =
        consumers   = 0;
        lastReturn  = CurrentTime();
    }

// ---------------------------------------------------------------------------

unsigned long Bin::Producers() const
    {
        return producers;
    }

// ---------------------------------------------------------------------------

unsigned long Bin::Consumers() const
    {
        return consumers;
    }

// ---------------------------------------------------------------------------

unsigned long Bin::Initial() const
    {
        return initial;
    }

// ---------------------------------------------------------------------------

unsigned long Bin::Maximum() const
    {
        return maximum;
    }

// ---------------------------------------------------------------------------

unsigned long Bin::Avail() const
    {
        return avail;
    }

// ---------------------------------------------------------------------------

double Bin::AvgAvail () const
    {
        const char* where = "Bin::Avail";

        if (!valid (className, where))
            return Undefined;

        SimTime now     = CurrentTime();
        SimTime diff    = now - ResetAt();
        double  wSumAvl = wSumAvail + avail * (now - lastReturn).Time();

        if (diff < Epsilon())
        {
            Warning (   "DivByZero", where, "-1.0 returned");
            return Undefined;
        }

        return wSumAvl / diff.Time();
```

```
    }

// --------------------------------------------------------------------------

Reporter* Bin::NewReporter () const
    {
        return new BinReporter (*this);
    }

// --------------------------------------------------------------------------

String Bin::ClassName () const
    {
        return className;
    }

// --------------------------------------------------------------------------
```

booldist.h

```
// --------------------------------------------------------------------------
//
// Datei
//          booldist.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// --------------------------------------------------------------------------

#ifndef BOOLDIST_H
#define BOOLDIST_H

#include "distribu.h"   // Basisklasse

#include "boolean.h"
#include "str.h"

// --------------------------------------------------------------------------

class BoolDist : public Distribution
{
    public:
        virtual bool        Sample () = 0;

        virtual     ~BoolDist ();
    protected:
                    BoolDist (  Model& owner,
                                const String& name = "",
                                bool showInReport = true,
                                bool showInTrace = false);
};

// --------------------------------------------------------------------------
// --------------------------------------------------------------------------

class BoolDistConst : public BoolDist
{
    public:
                    BoolDistConst ( Model& owner,
                                    const String& name = "",
                                    bool value = true,
                                    bool showInReport = true,
                                    bool showInTrace = false);
        virtual    ~BoolDistConst ();

        virtual bool        Sample ();
        virtual String      GetType() const;
                                //  liefert die Typ-Bezeichnung des ZZ-Stroms
```

```
                bool        GetValue() const;
                void        ChangeParameter (bool newValue);
        virtual Reporter*   NewReporter() const;
    private:
                bool        value;
};

// --------------------------------------------------------------------------

class BoolDistBernoulli : public BoolDist
{
    public:
                    BoolDistBernoulli ( Model& owner,
                                        const String& name = "",
                                        double probability = 0.0,
                                        bool showInReport = true,
                                        bool showInTrace = false);
        virtual    ~BoolDistBernoulli ();

        virtual bool        Sample ();
        virtual String      GetType() const;
                                // liefert die Typ-Bezeichnung des ZZ-Stroms
                double      GetProbability() const;
                void        ChangeParameter (double newProb);
        virtual Reporter*   NewReporter() const;
    protected:
                void        checkProb (const char* where);
    private:
                double      prob;       // Wahrscheinlichkeit fuer true
};

// --------------------------------------------------------------------------

#endif  // BOOLDIST_H
```

booldist.cc

```
// --------------------------------------------------------------------------
//
// Datei
//          booldist.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// --------------------------------------------------------------------------

#include "booldist.h"

#include "msgdist.h"
#include "repdist.h"

// --------------------------------------------------------------------------

static const char* className = "BoolDist";

// --------------------------------------------------------------------------

BoolDist::BoolDist (    Model& owner,
                        const String& name,
                        bool showInReport,
                        bool showInTrace)
    :   Distribution(owner, name, showInReport, showInTrace)
    {}

// --------------------------------------------------------------------------

BoolDist::~BoolDist ()
    {}
```

```
// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

BoolDistConst::BoolDistConst (  Model& owner,
                                const String& name,
                                bool Value,
                                bool showInReport,
                                bool showInTrace)
    :   BoolDist(owner, name, showInReport, showInTrace),
        value(Value)
    {
        state = Distribution::Initialized;
    }

// ---------------------------------------------------------------------------

BoolDistConst::~BoolDistConst ()
    {}

// ---------------------------------------------------------------------------

bool BoolDistConst::Sample ()
    {
        const char* where = "BoolDistConst::Sample";
        if (!valid (className, where))
            return false;
        checkSample (where);
        IncObservations();
        if (TraceIsOn())
            SendMessage (TrcDistBSample (*this, value));
        return value;
    }

// ---------------------------------------------------------------------------

String BoolDistConst::GetType () const
    {
        return "B-Constant";
    }

// ---------------------------------------------------------------------------

bool BoolDistConst::GetValue () const
    {
        return value;
    }

// ---------------------------------------------------------------------------

void BoolDistConst::ChangeParameter (bool newValue)
    {
        const char* where = "BoolDistConst::ChangeParameter";
        if (checkParam (where))
            value = newValue;
    }

// ---------------------------------------------------------------------------

Reporter* BoolDistConst::NewReporter () const
    {
        return new BoolDistConstReporter (*this);
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

BoolDistBernoulli::BoolDistBernoulli (  Model& owner,
                                        const String& name,
                                        double probability,
                                        bool showInReport,
                                        bool showInTrace)
    :   BoolDist(owner, name, showInReport, showInTrace),
        prob(probability)
    {
        state = Distribution::Initialized;
    }

// ---------------------------------------------------------------------------

BoolDistBernoulli::~BoolDistBernoulli ()
    {}

// ---------------------------------------------------------------------------
```

```
void BoolDistBernoulli::checkProb (const char* where)
    {
        if (prob < 0.0)
        {
            SendMessage (MsgDistProbNeg (where, *this, prob));
            prob = 0.0;
        } else
            if (prob > 1.0)
            {
                SendMessage (MsgDistProbTooBig (where, *this, prob));
                prob = 1.0;
            }
    }

// ------------------------------------------------------------------------------

bool BoolDistBernoulli::Sample ()
    {
        const char* where = "BoolDistBernoulli::Sample";
        if (!valid (className, where))
            return false;
        checkSample (where);
        IncObservations();
        bool b = (prob > random()) ? true : false;
        if (TraceIsOn())
            SendMessage (TrcDistBSample (*this, b));
        return b;
    }

// ------------------------------------------------------------------------------

String BoolDistBernoulli::GetType () const
    {
        return "Bernoulli";
    }

// ------------------------------------------------------------------------------

double BoolDistBernoulli::GetProbability () const
    {
        return prob;
    }

// ------------------------------------------------------------------------------

void BoolDistBernoulli::ChangeParameter (double newProb)
    {
        const char* where = "BoolDistBernoulli::ChangeParameter";
        if (checkParam (where))
        {
            prob = newProb;
            checkProb (where);
        }
    }

// ------------------------------------------------------------------------------

Reporter* BoolDistBernoulli::NewReporter () const
    {
        return new BoolDistBernoulliReporter (*this);
    }

// ------------------------------------------------------------------------------
```

boolean.h

```
// ------------------------------------------------------------------------------
//
// Datei
//          boolean.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
```

```
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------
//
// Beschreibung
//
//          Da C++ in manchen Implementatierungen den Typ bool nicht
//          unterstuetzt, wird hier ggf. mittels typedef ein
//          entsprechender Typ sowie die Konstanten true und
//          false definiert.
//
// ---------------------------------------------------------------------------

#ifndef  BOOLEAN_H
#define  BOOLEAN_H

// ---------------------------------------------------------------------------
// Unterstuetzt Metrowerks den Typ bool?

#ifdef __MWERKS__
#if !__option(bool)
#define NoBoolSupport
#endif  //  __option(bool)
#endif  // __MWERKS__

// ---------------------------------------------------------------------------

#ifdef NoBoolSupport    // wenn der Typ bool nicht unterstuetzt wird

typedef int bool
const bool true 1
const false 0

#endif  // NoBoolSupport

// ---------------------------------------------------------------------------

#endif //BOOLEAN_H
```

conditio.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          conditio.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#ifndef CONDITION_H
#define CONDITION_H

// ---------------------------------------------------------------------------

#include "modelcom.h"   // Basiklasse

#include "boolean.h"
#include "str.h"

// ---------------------------------------------------------------------------

class Model;
```

```
// --------------------------------------------------------------------------

class Condition : public ModelComponent
    /*  Bedingungen werden benoetigt, um bestimmte Entities zu finden.
        Hierzu wird in der Unterklasse die Methode Check definiert. Sie
        soll true liefern, wenn das uebergebene Entity die Bedingung
        erfuellt, andernfalls false.
    */
{
    public:
                    Condition (      Model&  owner,
                               const String& name = "Condition",
                                     bool    showInTrace = true);

        virtual bool    Check (const Entity& entity) const = 0;

                String  ClassName () const;
};

// --------------------------------------------------------------------------

#endif  // CONDITION_H
```

conditio.cc

```
// --------------------------------------------------------------------------
//
// Datei
//          conditio.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// --------------------------------------------------------------------------

#include "conditio.h"

#include "entity.h"
#include "model.h"

// --------------------------------------------------------------------------

const char* className = "Condition";

// --------------------------------------------------------------------------

Condition::Condition (Model& m, const String& name, bool trace)
    :   ModelComponent(m, name, trace)
    {}

// --------------------------------------------------------------------------

String Condition::ClassName () const
    {
        return className;
    }

// --------------------------------------------------------------------------
```

condq.h

```
// --------------------------------------------------------------------------
//
```

```
// Datei
//          condq.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ------------------------------------------------------------------------------

#ifndef CONDQUEUE_H
#define CONDQUEUE_H

// ------------------------------------------------------------------------------

#include "qbased.h" // Basisklasse

#include "boolean.h"
#include "conditio.h"   // Condition
#include "str.h"

// ------------------------------------------------------------------------------

class QueueImpl;

// ------------------------------------------------------------------------------

class CondQueue : public QueueBased
{
        CondQueue&  operator= (const CondQueue&);
                        // Zuweisung nicht implementiert

    public:
                    CondQueue ( Model&  owner,
                        const   String& name,
                                bool    checkAll = false,
                                bool    showInReport = true,
                                bool    showInTrace = true);
                    CondQueue (const CondQueue&);

        virtual    ~CondQueue ();

                void        WaitUntil (const Condition& c);
                                /*  Ist beim Aufruf die Bedingung erfuellt,
                                    so erfolgt keine Blockierung.
                                */
                void        Signal ();

                bool        CheckAll() const;

        virtual Reporter*   NewReporter() const;

                String      ClassName () const;

    protected:

    private:
                bool        checkProcess   (Process&        p,
                                            const    char*  where) const;
                void        activateAsNext (Process&        p,
                                            const    char*  where) const;

        QueueImpl&          qimpl;
        bool                checkAll;
};

// ------------------------------------------------------------------------------

#endif  // CONDQUEUE_H
```

condq.cc

```
// --------------------------------------------------------------------------
//
// Datei
//          condq.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// --------------------------------------------------------------------------

#include "condq.h"

#include "pblocker.h"
#include "process.h"
#include "qimpl.h"
#include "repcondq.h"
#include "msgcondq.h"

#include <assert.h>

// --------------------------------------------------------------------------

static const char* className = "CondQueue";

// --------------------------------------------------------------------------

CondQueue::CondQueue (  Model&  owner,
                const   String& name,
                        bool    CheckAll,
                        bool    showInReport,
                        bool    showInTrace)
    :   QueueBased  (owner, name, showInReport, showInTrace),
        qimpl       (*new QueueImpl (*this)),
        checkAll    (CheckAll)
    {}

// --------------------------------------------------------------------------

CondQueue::CondQueue (const CondQueue& cq)
    :   QueueBased  (cq),
        qimpl       (*new QueueImpl (*this)),
        checkAll    (cq.checkAll)
    {}

// --------------------------------------------------------------------------

CondQueue::~CondQueue ()
    {
        delete &qimpl;
    }

// --------------------------------------------------------------------------

bool CondQueue::checkProcess (Process& p, const char* where) const
    {
        if (!p.Valid ())
        {
            Warning ("invalid object", where);
            return false;
        }
        if (p.IsNullProcess())
        {
            Warning ("only processes may wait in a CondQueue",
                     where);
            return false;
        }
        if (!IsExperimentCompatible (p))
        {
            Warning ("attemp to mix components of different experiments",
                     where, "ignored");
            return false;
```

```
        }
        if (!IsModelCompatible (p))
        {
            Warning ("incompatible process", where);
            return false;
        }
        return true;
    }

// --------------------------------------------------------------------------

void CondQueue::activateAsNext (Process& process, const char* where) const
    {
        if (!process.IsNullProcess())
        {
            if (!checkProcess (process, where))
                return;

            if (process.IsScheduled())
            {   // anders als in DESMO!
                process.SkipTraceNote ();
                process.Cancel();
            }

            bool wasBlocked = process.Blocked();
            if (wasBlocked)
                ProcessBlocker::UnBlock (process);
                    // um Aktivierung zu erlauben

            process.SkipTraceNote ();
            process.ActivateAfter (Current());

            if (wasBlocked) ProcessBlocker::SetBlocked (process);
        }
    }

// --------------------------------------------------------------------------

void CondQueue::WaitUntil (const Condition& cond)
    {
        const char*     where = "CondQueue::WaitUntil";

        if (!valid (className, where))
            return;

        Process& process = CurrentProcess();
        if (!checkProcess (process, where))
            return;

        if (!valid (cond, "Condition", where))
            return;
        if (!IsExperimentCompatible (cond))
        {
            Warning ("attempt to mix components of different experiments",
                     where, "ignored");
            return;
        }
        if (!IsModelCompatible (cond))
        {
            Warning (   "incompatible Condition",
                        where,
                        "call is beeing ignored");
            return;
        }

        qimpl.Insert (process, where);  // in implizite Queue einreihen

        if (!cond.Check (process))  // Bedingung nicht erfuellt
        {
            if (TraceIsOn())
                SendMessage (TrcCQWaitUntil (*this, cond));

            bool resume = false;
            do
            {
                ProcessBlocker::Block (process);

                resume = cond.Check (process);
                if (checkAll || resume)
                    activateAsNext ((Process&)qimpl.Succ (process, where),
                                    where);
            } while (!resume);

            if (TraceIsOn())
```

```
                SendMessage (TrcCQLeave (*this, cond));
        }

        qimpl.Remove (process, where);  // aus impliziter Queue entfernen
        ProcessBlocker::UnBlock (process);
    }

// ----------------------------------------------------------------------------

void CondQueue::Signal()
    {
        const char* where = "CondQueue::Signal";

        if (!valid (className, where))
            return;

        if (TraceIsOn())
            SendMessage (TrcCQSignal (*this));

        activateAsNext ((Process&)qimpl.First (where), where);
    }

// ----------------------------------------------------------------------------

bool CondQueue::CheckAll() const
    {
        return checkAll;
    }

// ----------------------------------------------------------------------------

Reporter* CondQueue::NewReporter () const
    {
        return new CondQueueReporter (*this);
    }

// ----------------------------------------------------------------------------

String CondQueue::ClassName () const
    {
        return className;
    }

// ----------------------------------------------------------------------------
```

coop.h

```
// ----------------------------------------------------------------------------
//
// Datei
//          coop.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ----------------------------------------------------------------------------

#ifndef PROCESSCOOPERATION_H
#define PROCESSCOOPERATION_H

// ----------------------------------------------------------------------------

#include "dyobject.h"   // Basisklasse
#include "process.h"    // Friend-Klasse

#include "simtime.h"

// ----------------------------------------------------------------------------
```

```
class Model;
class InterruptCode;

// ---------------------------------------------------------------------------

class ProcessCooperation : public DynamicalObject
    /*  ProcessCooperation dient als Oberklasse fuer Objekte, die eine
        Kooperation zwischen zwei Prozessen repraesentieren. Sobald
        eine Kooperation zustande kommt, wird die Methode Cooperation ()
        gerufen, die in abgeleiteten Klassen definiert werden muss, um
        die gemeinsamen Handlungen zu beschreiben. Waehrend der Kooperation
        gilt der Master als aktiv, der Slave als passiv. */
    /*  Nach Beendigung der Kooperation wird zuerst der Master und
        anschliessend der Slave angestossen, falls dieser nicht waehrend
        der Kooperation aktiviert wurde.
    */
{
        friend class Process;   // Setzt master und slave und ruft Cooperate()

    public:
                    ProcessCooperation (  Model&  owner,
                                    const String& name = "",
                                          bool    showInTrace = true);
        virtual    ~ProcessCooperation ();

    protected:
        virtual void            Cooperation (Process& master,
                                             Process& slave) = 0;
                                    /*  Gemeinsame Handlungen von Master und
                                        Slave */

                // die folgenden Methoden werden an den Master weitergeleitet
                void            Activate (SimTime dt);
                                    /*  Vormerken zum Zeitpunkt now + dt */
                void            ReActivate (SimTime dt);
                                    /*  analog zu ReSchedule */
                void            ActivateBefore (Schedulable& before);

                void            ActivateAfter (Schedulable& after);

                void            Hold (SimTime dt);
                                    /*  Reaktivierung bei now + dt.
                                        Konzeptionell hat der Prozess hier
                                        seine aktive zeitverbrauchende Phase.
                                        */
                void            Passivate ();
                                    /*  Passivierung fuer unbestimmte Zeit.
                                        Der Prozess kann nur noch durch andere
                                        Objekte wieder aktiviert werden. */

                bool            Interrupted () const;
                InterruptCode   GetInterruptCode () const;
                void            ClearInterruptCode ();

                void            Schedule (SimTime dt, Event& ev);
                                    /*  Vormerken des Masters mit dem
                                        Ereignis ev zum Zeitpunkt
                                        now + dt. */

                void            ScheduleBefore (Schedulable& before,
                                                Event&       ev);

                void            ScheduleAfter (Schedulable& after, Event& ev);

                PriorityT       GetPriority () const;
                                    /*  Abfragen der Prioritaet des Entities
                                        je kleiner die Prioritaet, desto mehr
                                        wird das Entity beim Einreihen in
                                        Warteschlangen bevorzugt */
                PriorityT       SetPriority (const PriorityT newPriority);
                                    /*  Setzen der Prioritaet. Sie hat
                                        Einfluss auf die Rangfolge in allen
                                        im- und expliziten Warteschlangen.
                                        Die neue Prioritaet wird
                                        zurueckgegeben. */

                QueueOption     GetQueueOption() const;

                Event&          NextEvent () const;
                                    // Das naechste vorgemerkte Ereignis
                Entity&         NextEntity () const;    // DESMO: NextEv ()
                                    // Das naechste vorgemerkte Entity
                Process&        NextProcess () const;
                                    // Der naechste vorgemerkte Prozess
```

```
                String          ClassName () const;

    private:
        Process&    currentProcess (const char* where,
                                    const char* conseq = "ignored") const;
};

// ---------------------------------------------------------------------------

#endif  // PROCESSCOOPERATION_H
```

coop.cc

```
// ---------------------------------------------------------------------------
//
// Datei
//          coop.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#include "coop.h"

#include "model.h"
#include "process.h"

// ---------------------------------------------------------------------------

static const char* className = "ProcessCooperation";

// ---------------------------------------------------------------------------

ProcessCooperation::ProcessCooperation (Model& owner, const String& name,
                                        bool showInTrace)
    :   DynamicalObject (owner, name, showInTrace)
    {}

// ---------------------------------------------------------------------------

ProcessCooperation::~ProcessCooperation ()
    {}

// ---------------------------------------------------------------------------

Process& ProcessCooperation::currentProcess (const char* where,
                                             const char* conseq) const
    {
        Process& p = CurrentProcess();
        if (p.IsNullProcess())
            Warning (   "master is not current", where, conseq);
        return p;
    }

// ---------------------------------------------------------------------------

void ProcessCooperation::Activate (SimTime dt)
    {
        const char* where = "ProcessCooperation::Activate";
        Process& p = currentProcess (where);

        if (!p.IsNullProcess())
            p.Activate (dt);
    }

// ---------------------------------------------------------------------------
```

```
void ProcessCooperation::ReActivate (SimTime dt)
    {
        const char* where = "ProcessCooperation::ReActivate";
        Process& p = currentProcess (where);

        if (!p.IsNullProcess())
            p.ReActivate (dt);
    }

// ---------------------------------------------------------------------------

void ProcessCooperation::ActivateBefore (Schedulable& before)
    {
        const char* where = "ProcessCooperation::ActivateBefore";
        Process& p = currentProcess (where);

        if (!p.IsNullProcess())
            p.ActivateBefore (before);
    }

// ---------------------------------------------------------------------------

void ProcessCooperation::ActivateAfter (Schedulable& after)
    {
        const char* where = "ProcessCooperation::ActivateAfter";
        Process& p = currentProcess (where);

        if (!p.IsNullProcess())
            p.ActivateAfter (after);
    }

// ---------------------------------------------------------------------------

void ProcessCooperation::Hold (SimTime dt)
    {
        const char* where = "ProcessCooperation::Hold";
        Process& p = currentProcess (where);

        if (!p.IsNullProcess())
            p.Hold (dt);
    }

// ---------------------------------------------------------------------------

void ProcessCooperation::Passivate ()
    {
        const char* where = "ProcessCooperation::";
        Process& p = currentProcess (where);

        if (!p.IsNullProcess())
        CurrentProcess().Passivate ();
    }

// ---------------------------------------------------------------------------

bool ProcessCooperation::Interrupted () const
    {
        const char* where = "ProcessCooperation::Interrupted";
        Process& p = currentProcess (where, "false is returned");

        if (!p.IsNullProcess())
            return p.Interrupted ();
        else
            return false;
    }

// ---------------------------------------------------------------------------

InterruptCode ProcessCooperation::GetInterruptCode () const
    {
        const char* where = "ProcessCooperation::GetInterruptCode";
        Process& p = currentProcess (where, "'NoInterrupt' is returned");

        if (!p.IsNullProcess())
            return p.GetInterruptCode ();
        else
            return InterruptCode::NoInterrupt();
    }

// ---------------------------------------------------------------------------

void ProcessCooperation::ClearInterruptCode ()
    {
        const char* where = "ProcessCooperation::ClearInterruptCode";
```

```
        Process& p = currentProcess (where);

        if (!p.IsNullProcess())
            p.ClearInterruptCode ();
    }

// ---------------------------------------------------------------------------

void ProcessCooperation::Schedule (SimTime dt, Event& ev)
    {
        const char* where = "ProcessCooperation::Schedule";
        Process& p = currentProcess (where);

        if (!p.IsNullProcess())
            p.Schedule (dt, ev);
    }

// ---------------------------------------------------------------------------

void ProcessCooperation::ScheduleBefore (Schedulable& before, Event& ev)
    {
        const char* where = "ProcessCooperation::ScheduleBefore";
        Process& p = currentProcess (where);

        if (!p.IsNullProcess())
            p.ScheduleBefore (before, ev);
    }

// ---------------------------------------------------------------------------

void ProcessCooperation::ScheduleAfter (Schedulable& after, Event& ev)
    {
        const char* where = "ProcessCooperation::ScheduleAfter";
        Process& p = currentProcess (where);

        if (!p.IsNullProcess())
            p.ScheduleAfter (after, ev);
    }

// ---------------------------------------------------------------------------

PriorityT ProcessCooperation::GetPriority () const
    {
        const char* where = "ProcessCooperation::GetPriority";
        Process& p = currentProcess (where, "0 is returned");

        if (!p.IsNullProcess())
            return p.GetPriority ();
        else
            return 0;
    }

// ---------------------------------------------------------------------------

PriorityT ProcessCooperation::SetPriority (const PriorityT newPriority)
    {
        const char* where = "ProcessCooperation::SetPriority";
        Process& p = currentProcess (where, "ignored and 0 is returned");

        if (!p.IsNullProcess())
            return p.SetPriority (newPriority);
        else
            return 0;
    }

// ---------------------------------------------------------------------------

QueueOption ProcessCooperation::GetQueueOption() const
    {
        const char* where = "ProcessCooperation::GetQueueOption";
        Process& p = currentProcess (where, "'OnlyOneQueue' is returned");

        if (!p.IsNullProcess())
            return p.GetQueueOption ();
        else
            return OnlyOneQueue;
    }

// ---------------------------------------------------------------------------

Event& ProcessCooperation::NextEvent() const
    {
        const char* where = "ProcessCooperation::NextEvent";
        Process& p = currentProcess (where, "NullEvent is returned");
```

```
        if (!p.IsNullProcess())
            return p.NextEvent ();
        else
            return NullEvent();
    }

// ---------------------------------------------------------------------------

Entity& ProcessCooperation::NextEntity() const
    {
        const char* where = "ProcessCooperation::NextEntity";
        Process& p = currentProcess (where, "NullEntity is returned");

        if (!p.IsNullProcess())
            return p.NextEntity ();
        else
            return NullEntity();
    }

// ---------------------------------------------------------------------------

Process& ProcessCooperation::NextProcess() const
    {
        const char* where = "ProcessCooperation::NextProcess";
        Process& p = currentProcess (where, "NullProcess is returned");

        if (!p.IsNullProcess())
            return p.NextProcess ();
        else
            return NullProcess();
    }

// ---------------------------------------------------------------------------

String ProcessCooperation::ClassName () const
    {
        return className;
    }

// ---------------------------------------------------------------------------
```

coroutin.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          coroutin.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

// Version 04
#ifndef COROUTINE_H
#define COROUTINE_H

// ---------------------------------------------------------------------------

#include <setjmp.h>
#include <iostream.h>

// ---------------------------------------------------------------------------

class Coroutine
{
    public:
                                Coroutine ();
```

```
        virtual            ~Coroutine ();

                void        Transfer ();

        static  Coroutine*  MainCoroutine ();
        static  Coroutine*  ActiveCoroutine ();
        static  Coroutine*  LastCoroutine ();

                void        Debug () const;     // gibt Infos aus
        static  void        SetStreams (ostream& outStream,
                                        ostream& errStream);
        static  void        ResetStreams ();
    protected:
                long        Id () const;

    private:
                            Coroutine (const Coroutine&);
        Coroutine&          operator= (const Coroutine&);
        // Copy-Konstruktor und Zuweisung sollen nicht benutzt werden, da
        // keine sinnvolle bzw. sichere Semantik existiert. Sie sind nicht
        // implementiert, so dass spaetestens der Linker einen Fehler
        // meldet, wenn sie doch benutzt weurden

        virtual void        Body () = 0;
                            // die Aktionen der Koroutine

                int         Valid () const;     // liefert bool
        static  void        RestoreStack ();
        static  char*       StackPointer (int); // aktueller Stackpointer
        static  int         StackGrowsUp ();    // boolean
        static  void        InitMainCoroutine ();

        static  ostream*    out;
        static  ostream*    err;
        static  Coroutine*  mainCoroutine;
        static  Coroutine*  activeCoroutine;
        static  Coroutine*  lastCoroutine;
        static  int         initialized;        // bool
        static  int         stackGrowsUp;       // Stackrichtung (bool)
        static  char*       coroutineStackBase; // Beginn fuer neue Koroutinen
        static  jmp_buf     initialContext;     // fuer erste Ausfuehrung
        static  jmp_buf     copyContext;        // fuer RestoreStack()
        static  char*       copyContextSP;      // StackPointer fuer &dummy
        static  long        nextId;

        long        id;             // lfd. Nummer
        char*       lowerStackEnd;  // Kopieradresse im Stack
                                    // fuer memcpy()
        char*       stackCopy;      // die gesicherte Kopie
        long        size;           // Groesse der Kopie
        jmp_buf     context;        // fuer setjmp (Registerwerte)
        long        validationField;// muss stets = validationConstant sein!
};

// ---------------------------------------------------------------------------

#endif
```

coroutin.cc

```
// ---------------------------------------------------------------------------
//
// Datei
//          coroutin.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------
```

```
#include "coroutin.h"
#include <iostream.h>
#include <assert.h>
#include <stdlib.h>
#include <string.h>
#include <setjmp.h>

// ---------------------------------------------------------------------------

class MainProgram : public Coroutine
{
    private:
        void    Body ();    // darf nie aufgerufen werden
};

// ---------------------------------------------------------------------------

void MainProgram::Body ()
{
    assert ((1==0));        // ist auf jeden Fall ein Fehler!
}

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

const                  validationNumber         = 0x76912362L;

ostream*    Coroutine::out                      = &cout;
ostream*    Coroutine::err                      = &cerr;
int         Coroutine::initialized              = (1 != 1); // FALSE
int         Coroutine::stackGrowsUp             = 0;
char*       Coroutine::coroutineStackBase       = 0;
long        Coroutine::nextId                   = 0;
Coroutine*  Coroutine::mainCoroutine            = 0;
Coroutine*  Coroutine::lastCoroutine            = 0;
Coroutine*  Coroutine::activeCoroutine          = 0;
jmp_buf     Coroutine::initialContext           = {0};
jmp_buf     Coroutine::copyContext              = {0};
char*       Coroutine::copyContextSP            = 0;

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

Coroutine::Coroutine ()
    :   id (nextId++), lowerStackEnd (0), stackCopy (0), size (0),
        validationField (validationNumber)
{
    if (!initialized)   // dann ist dies die erste Coroutine, die erzeugt wird
        InitMainCoroutine ();

    // initialContext ist hier auf jeden Fall initialisiert
    memcpy (context, initialContext, sizeof (jmp_buf));
}

// ---------------------------------------------------------------------------

Coroutine::~Coroutine ()
{
    assert (Valid ());
    assert ((activeCoroutine != this) && (mainCoroutine != this));
    validationField = 0;
    if (stackCopy)  {   // da ist etwas freizugeben
        delete [] stackCopy;
        stackCopy = 0;
    }
}

// ---------------------------------------------------------------------------

void Coroutine::Debug () const
{
    assert (Valid ());
    *out << "Allgemeine Informationen:\n=========================\n";
    *out << "Stackrichtung (" << (1==1) << " = aufsteigend): "
                              << StackGrowsUp () << endl;
    *out << "Koroutinenbasis: " << (unsigned long) coroutineStackBase << endl;
    *out << "Naechste ID: " << nextId << endl;
    *out << "\nSpezielle Informationen:\n=========================\n";
    *out << "Id: " << id << endl;
    *out << "Groesse der Stackkopie: " << size << endl;
}

// ---------------------------------------------------------------------------
```

```
void Coroutine::SetStreams (ostream& outs, ostream& errs)
{
    out = &outs;
    err = &errs;
}

// ---------------------------------------------------------------------------

void Coroutine::ResetStreams ()
{
    out = &cout;
    err = &cerr;
}

// ---------------------------------------------------------------------------

char* Coroutine::StackPointer (int cnt) // darf auf keinen Fall inline sein!
{
#if defined(__GNUC__) && defined(sun)
    char dummy[20];
#else
    char dummy[1];
#endif
    char *ptr = dummy;
    // Warnung "pointer to local variable returned" moeglich, aber OK
    return (cnt > 0) ? StackPointer (--cnt) : ptr;
}

// ---------------------------------------------------------------------------

void Coroutine::InitMainCoroutine  ()
{
    char dummy;

    initialized         = (1==1);   // TRUE

    if (setjmp (initialContext) != 0) {
        // von hier werden all neuen Koroutinen gestartet (longjmp)
        activeCoroutine->Body ();
        // darf aus Body nie zurueckkerhren!
        *err<<  "Coroutine::Transfer illegal end of Coroutine procedure\n"
            <<  "\tencountered.\n"
            <<  "\tProgram will be halted.\n" << flush;
        exit(0);
    }
    else {
        // beim ersten und einzigen Aufruf von InitMainCoroutine
        stackGrowsUp        = StackPointer (0) > &dummy;
        coroutineStackBase  = &dummy;
        activeCoroutine =
        lastCoroutine   =
        mainCoroutine       = new MainProgram; assert (mainCoroutine);
        if (setjmp (mainCoroutine->context) == 0)
            RestoreStack ();    // initialisiere copyContext
    }
}

// ---------------------------------------------------------------------------

int Coroutine::StackGrowsUp  ()
{
    return stackGrowsUp;
}

// ---------------------------------------------------------------------------

Coroutine* Coroutine::MainCoroutine ()
{
    if (!initialized)
        InitMainCoroutine ();

    return mainCoroutine;
}

// ---------------------------------------------------------------------------

Coroutine* Coroutine::LastCoroutine ()
{
    if (!initialized)
        InitMainCoroutine ();

    return lastCoroutine;
}
```

```
// ---------------------------------------------------------------------------

Coroutine* Coroutine::ActiveCoroutine ()
{
    if (!initialized)
        InitMainCoroutine ();

    return activeCoroutine;
}

// ---------------------------------------------------------------------------

void Coroutine::RestoreStack ()
{
    const dummySize = 20;
    char dummy[dummySize];

    // stackpointer ueber dummy ermitteln und global verfuegbar machen
    if (stackGrowsUp)
        copyContextSP = &dummy[dummySize-1];
    else
        copyContextSP = &dummy[0];

    setjmp (copyContext);   // Zustand sichern,
                            // um Rekursionstiefe zu vermindern

    if (activeCoroutine->size > 0) {
        // da ist ein Stack zu restaurieren
        if (stackGrowsUp) {
            if (copyContextSP <   activeCoroutine->lowerStackEnd
                                + activeCoroutine->size)
                RestoreStack(); // Stackptr noch nicht im sicheren Bereich
        } else {
            if (activeCoroutine->lowerStackEnd < copyContextSP)
                RestoreStack(); // Stackptr noch nicht im sicheren Bereich
        }
        memcpy(activeCoroutine->lowerStackEnd,
               activeCoroutine->stackCopy, activeCoroutine->size);
        delete[] activeCoroutine->stackCopy;
        activeCoroutine->stackCopy  = 0;
        activeCoroutine->size       = 0;
    }
    longjmp (activeCoroutine->context,1);   // Kontext herstellen
}

// ---------------------------------------------------------------------------

void Coroutine::Transfer ()
{
#if defined(__GNUC__) && defined(sun)
    const stkCount = 10;
#else
    const stkCount =  0;
#endif

    assert (Valid ());
    assert (activeCoroutine->Valid ());
    if (this == activeCoroutine) {
        *err << "Info from 'Coroutine::Transfer':\n"
                "\tControl remains in active Coroutine\n" << flush;
        return;
    }

    if (stackGrowsUp) {
        // Stack waechst nach oben
        activeCoroutine->size           = (long) (StackPointer(stkCount)
                                        - coroutineStackBase);
        activeCoroutine->lowerStackEnd  = coroutineStackBase;
    }
    else {
        // Stack waechst nach unten
        activeCoroutine->lowerStackEnd  = StackPointer(stkCount);
        activeCoroutine->size           = (long) (coroutineStackBase
                                        - activeCoroutine->lowerStackEnd);
    }

    // den Stack sichern
    if (activeCoroutine->size > 0)
    {
        activeCoroutine->stackCopy = new char [activeCoroutine->size];
        assert(activeCoroutine->stackCopy != 0);
        memcpy(activeCoroutine->stackCopy, activeCoroutine->lowerStackEnd,
               activeCoroutine->size);
```

```
    }

    lastCoroutine   = activeCoroutine;
    activeCoroutine = this;

    // den Kontext sichern
    if (setjmp (lastCoroutine->context) == 0) { // damit ist Zustand gesichert
        // hier geht es weiter, wenn wir nicht ueber longjmp kommen
        // Stack kopieren (in RestoreStack()) und
        // Stackpointer fuer die neue (jetzt aktive) setzen
        longjmp (copyContext,1);
    }
}

// ----------------------------------------------------------------------------

long Coroutine::Id () const
{
    assert (Valid ());
    return id;
}

// ----------------------------------------------------------------------------

int Coroutine::Valid () const
{
    return (validationField == validationNumber);
}

// ----------------------------------------------------------------------------
```

count.h

```
// ----------------------------------------------------------------------------
//
// Datei
//          count.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ----------------------------------------------------------------------------

#ifndef COUNT_H
#define COUNT_H

// ----------------------------------------------------------------------------

#include "statobj.h"    // Basisklasse

#include "str.h"

// ----------------------------------------------------------------------------

class Reporter;

// ----------------------------------------------------------------------------

class Count : public StatisticObject
{
    public:
                    Count (         Model& owner,
                            const   String& name = "",
                            bool    showInReport = true,
                            bool    showInTrace  = false);

        virtual void            Update ();
                void            Update (unsigned long c);
                unsigned long   Value() const;
```

```
        virtual Reporter*       NewReporter() const;

                String          ClassName () const;
};

// ---------------------------------------------------------------------------

#endif  // COUNT_H
```

count.cc

```
// ---------------------------------------------------------------------------
//
// Datei
//          count.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#include "count.h"

#include "repstat.h"

// ---------------------------------------------------------------------------

static const char* className = "Count";

// ---------------------------------------------------------------------------

Count::Count (           Model& owner,
                const    String& name,
                         bool showInReport,
                         bool showInTrace)
    :   StatisticObject(owner, name, showInReport, showInTrace)
    {}

// ---------------------------------------------------------------------------

void Count::Update ()
    {
        Update (1);
    }

// ---------------------------------------------------------------------------

void Count::Update (unsigned long c)
    {
        const char* where = "Count::Update";

        if (!valid (className, where))
            return;

        IncObservations (c);
        traceUpdate();
    }

// ---------------------------------------------------------------------------

unsigned long Count::Value () const
    {
        return Reportable::Observations();
    }

// ---------------------------------------------------------------------------

Reporter* Count::NewReporter () const
```

```
    {
        return new CountReporter (*this);
    }

// ---------------------------------------------------------------------------

String Count::ClassName () const
    {
        return className;
    }

// ---------------------------------------------------------------------------
```

distman.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          distman.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#ifndef DISTRIBMANAGER_H
#define DISTRIBMANAGER_H

// ---------------------------------------------------------------------------

#include "distribu.h"

#include "str.h"

// ---------------------------------------------------------------------------

class DistributionList;

// ---------------------------------------------------------------------------

class DistribManager
{
    public:
                    DistribManager ();
                   ~DistribManager ();

            void    Register (Distribution&);
            void    DeRegister (Distribution&);

            long    NextSeed();
                        // liefert den naechsten Startwert
            void    AntitheticAll();
                        // setzt alle Verteilungen auf 'antithetisch'
            void    ResetAll() const;
                        // setzt alle Verteilungen zurueck
            void    NewSeedAll ();
                        /* gibt allen Zufallszahlenstroemen einen neuen
                           Sartwert */

            double  Random (long& seed) const;
                        /*  berechnet basierend auf seed die naechste
                            Zufallszahl zwischen 0 und 1 */
            void    SetSeed (long& seed, long newSeed) const;
                        /*  setzt seed auf newSeed, bei 0 wird ein
                            spezieller Startwert eingesetzt */
            void    SeedGenerator (long newSeed);
                        /*  setzt den Startwert des Startwertegenerators auf
                            newSeed, bei 0 wird ein spezieller Startwert
                            eingesetzt */
```

```
    protected:

    private:
                    DistribManager (const DistribManager& objToCopy);
                        // Kopierkonstruktor nicht erlaubt
        DistribManager& operator= (const DistribManager&);
                        // Zuweisung nicht erlaubt

            long    Modulus() const;
                        // Periodenlaenge des Generators

            void    Multiply (long& seed, int k) const;
                        // Hilfsfunktion fuer Berechnungen

        DistributionList&   distributionList;
        long                currentSeed;  // aktueller 'Seed' des Startwert-
                                          // generators
        bool                antithetic;   // true, wenn Atithetic()
                                          // aufgerufen wurde alle von nun
                                          // an erzeugten Verteilungen
                                          // werden auf antithetisch gesetzt
};

// ---------------------------------------------------------------------------

#endif  // DISTRIBMANAGER_H
```

distman.cc

```
// ---------------------------------------------------------------------------
//
// Datei
//          distman.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#include "distman.h"

#include "distribu.h"
#include "ring.h"

// ---------------------------------------------------------------------------
// lokale Konstante

static const long cModulus = 67099547L;     // Periodenlaenge des Generators

// ---------------------------------------------------------------------------

class DistributionList : public Ring<Distribution>
{};

// ---------------------------------------------------------------------------

DistribManager::DistribManager ()
    :   distributionList (*new DistributionList),
        currentSeed (907),
        antithetic(false)
    {}

// ---------------------------------------------------------------------------

DistribManager::~DistribManager ()
    {
        delete &distributionList;
    }
```

```
// ---------------------------------------------------------------------------

void DistribManager::Register (Distribution& d)
    {
        // wie Stack behandeln, da Destruktoren wahrscheinlich in
        // umgekehrter Reihenfolge aufgerufen werden
        distributionList.Push (&d);
        if (antithetic)
            d.Antithetic();
    }

// ---------------------------------------------------------------------------

void DistribManager::DeRegister (Distribution& d)
    {
        distributionList.Remove (&d);
    }

// ---------------------------------------------------------------------------

inline long DistribManager::Modulus () const
    {
        return cModulus;
    }

// ---------------------------------------------------------------------------

inline void DistribManager::Multiply (long& seed, int k) const
    {
        seed = (seed * k) % Modulus();
    }

// ---------------------------------------------------------------------------

double DistribManager::Random (long& seed) const
    {
        Multiply (seed, 32);    // 8192 = 32 * 32 * 8
        Multiply (seed, 32);
        Multiply (seed,  8);
        return double (seed) / double (Modulus());
    }

// ---------------------------------------------------------------------------

void DistribManager::SetSeed (long& seed, long newSeed) const
    {
        if (newSeed == 0)
            seed = Modulus() / 2;
        else
            seed = newSeed % Modulus();
    }

// ---------------------------------------------------------------------------

void DistribManager::SeedGenerator (long newSeed)
    {
        SetSeed (currentSeed, newSeed);
    }

// ---------------------------------------------------------------------------

long DistribManager::NextSeed ()
// Hilfsfunktion: Erzeugung des naechsten Startgenerator-'Seed'-Wertes
// U (k+1) := (8192^120633 * U (k)) MOD 67009547
// 8192^120633 = 36855 (mod 67009547)
// Multiplikation portionsweise, um Ueberlauf zu verhindern.
    {
        Multiply (currentSeed,  7);     // 36855 = 7 * 13 * 15 * 27
        Multiply (currentSeed, 13);
        Multiply (currentSeed, 15);
        Multiply (currentSeed, 27);

        return currentSeed;
    }

// ---------------------------------------------------------------------------

void DistribManager::AntitheticAll ()
    {
        antithetic = true;
        Distribution*   d = distributionList.First ();
        for (int i = distributionList.Size (); i > 0; --i)
        {
            d->Antithetic();
```

```
            d = distributionList.Next();
        }
    }

// ---------------------------------------------------------------------------

void DistribManager::ResetAll () const
    {
        Distribution*   d = distributionList.First ();
        for (int i = distributionList.Size (); i > 0; --i)
        {
            d->Reset();
            d = distributionList.Next();
        }
    }

// ---------------------------------------------------------------------------

void DistribManager::NewSeedAll ()
    {
        Distribution*   d = distributionList.First ();
        for (int i = distributionList.Size (); i > 0; --i)
        {
            d->SetSeed (NextSeed());
            d = distributionList.Next();
        }
    }

// ---------------------------------------------------------------------------
```

distribu.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          distribu.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#ifndef DISTRIBUTION_H
#define DISTRIBUTION_H

// ---------------------------------------------------------------------------

#include "reportab.h"   // Basisklasse

#include "boolean.h"
#include "str.h"

// ---------------------------------------------------------------------------

class Distribution : public Reportable
{
        friend class DistribManager;
    public:
        virtual    ~Distribution ();

                long        Seed() const;
                                /*  liefert den Startwert des ZZ-Stroms */
        virtual String      GetType() const = 0;
                                /*  liefert die Typ-Bezeichnung des
                                    ZZ-Stroms als String */
                void        SetSeed (long newSeed);
                                /*  setzt den Startwert auf newSeed */
                void        SeedGenerator (long newSeed) const;
                                /*  setzt den Startwert des
```

```
                              Startwertegenerators auf newSeed */
                void        Antithetic();
                                /*  setzt den ZZ-Strom auf 'antithetisch'.
                                    Dabei wird auch der Startwert
                                    zurueckgesetzt. */
                void        AntitheticAll();
                                /*  setzt alle zum selben Experiment
                                    gehoerenden ZZ-Stroeme auf 'antithetisch'.
                                    Dabei werden auch die Startwerte
                                    zurueckgesetzt. */
                void        ResetAll();
                                /*  setzt alle zum selben Experiment
                                    gehoerenden ZZ-Stroeme zurueck. */
                String      ClassName () const;

    protected:
                    Distribution (Model& owner, const String& name = "",
                                  bool ShowInReport = true,
                                  bool showInTrace  = false);
                    Distribution (const Distribution& objToCopy);

        enum State_e {  Uninitialized,  // kann noch nicht sampeln
                        Initialized,    // kann benutzt werden
                        Used            // wurde bereits benutzt,
                     }                  // Parameter koennen nicht mehr
                                        // geaendert werden
                            state;

                double      random ();

                void        noInitError (const char* where);
                void        inUseError  (const char* where);
                void        deletedError(const char* where);
                void        checkSample (const char* where);
                bool        checkParam  (const char* where);

    private:
        Distribution&       operator= (const Distribution&);
                                // darf nicht benutzt werden
                                // nicht implementiert
        DistribManager&     distribManager; // enthaelt den Startwertgenerator

                long        initialSeed,    // Anfangsstartwert
                            seed;           // aktueller Startwert
                bool        antithetic;     // antithetisch J/N
};

// ---------------------------------------------------------------------------

#endif  // DISTRIBUTION_H
```

distribu.cc

```
// ---------------------------------------------------------------------------
//
// Datei
//          distribu.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#include "distribu.h"

#include "experimm.h"
#include "distman.h"
#include "msgdist.h"
```

```
#include <assert.h>

// --------------------------------------------------------------------------

static const char* className = "Distribution";

// --------------------------------------------------------------------------

Distribution::Distribution (Model& owner, const String& name,
                            bool rep, bool showInTrace)
    :   Reportable      (owner, name, rep, showInTrace),
        state           (Uninitialized),
        distribManager  (ExperimentManager::Instance().
                            GetDistribManager (*this)),
        initialSeed     (distribManager.NextSeed()),
        seed            (initialSeed),
        antithetic      (false)
    {
        distribManager.Register (*this);
    }

// --------------------------------------------------------------------------

Distribution::Distribution (const Distribution& d)
    :   Reportable      (d),
        state           (d.state),
        distribManager  (d.distribManager),
        initialSeed     (d.initialSeed),
        seed            (d.seed),
        antithetic      (d.antithetic)
    {
        const char* where = "Distribution::Distribution";
        if (!Valid())
        {
            // FatalError: Kopieren eines bereits geloeschten ZZ-Stroms
            SendMessage (MsgDistCopyInvalid(where, *this));
            assert (false);
        }
        if (state == Used)
            // Warnung: Kopieren eines bereits benutzten ZZ-Stroms
            SendMessage (MsgDistCopyUsed(where, *this));
        distribManager.Register (*this);
    }

// --------------------------------------------------------------------------

Distribution::~Distribution ()
    {
        distribManager.DeRegister (*this);
    }

// --------------------------------------------------------------------------

void Distribution::Antithetic()
    {
        const char* where = "Distribution::Antithetic";

        if (!valid (className, where))
            return;

        antithetic = true;
        seed = initialSeed;
    }

// --------------------------------------------------------------------------

void Distribution::AntitheticAll()
    {
        const char* where = "Distribution::AntitheticAll";

        if (!valid (className, where))
            return;

        distribManager.AntitheticAll();
    }

// --------------------------------------------------------------------------

void Distribution::ResetAll()
    {
        const char* where = "Distribution::ResetAll";

        if (!valid (className, where))
            return;
```

```
        distribManager.ResetAll();
    }

// ---------------------------------------------------------------------------

long Distribution::Seed() const
    {
        return initialSeed;
    }

// ---------------------------------------------------------------------------

void Distribution::SetSeed (long newSeed)
    {
        const char* where = "Distribution::SetSeed";

        if (!valid (className, where))
            return;

        distribManager.SetSeed (seed, newSeed);
        initialSeed = seed;
    }

// ---------------------------------------------------------------------------

void Distribution::SeedGenerator (long newSeed) const
    {
        const char* where = "Distribution::SeedGenerator";

        if (!valid (className, where))
            return;

        distribManager.SeedGenerator (newSeed);
    }

// ---------------------------------------------------------------------------

String Distribution::ClassName () const
    {
        return className;
    }

// ---------------------------------------------------------------------------

void Distribution::checkSample (const char* where)
    {
        if (!valid (className, where))
            return;

        if (state == Used)      // Normalfall, zuerst behandeln
            return;             // OK
        if (state == Uninitialized)
            noInitError (where);
        else
            state = Used;
    }

// ---------------------------------------------------------------------------

bool Distribution::checkParam (const char* where)
    {
        if (!valid (className, where))
            return false;

        bool ret = true;
        if (state == Used)
            { inUseError (where); ret = false; }
        else state = Initialized;
        return ret;
    }

// ---------------------------------------------------------------------------

double Distribution::random ()
    {
        if (antithetic)
            return 1.0 - distribManager.Random (seed);
        else
            return distribManager.Random (seed);
    }

// ---------------------------------------------------------------------------
```

```
void Distribution::noInitError (const char* where)
    {
        SendMessage (MsgDistNoInit (where, *this));
    }

// ---------------------------------------------------------------------------

void Distribution::inUseError (const char* where)
    {
        SendMessage (MsgDistChangeUsed (where, *this));
    }

// ---------------------------------------------------------------------------

void Distribution::deletedError (const char* where)
    {
        SendMessage (MsgDistUseDeleted (where));
    }

// ---------------------------------------------------------------------------
```

dyobjcat.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          dyobjcat.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#ifndef DYNOBJCATALOG_H
#define DYNOBJCATALOG_H

// ---------------------------------------------------------------------------

class DynamicalObject;
class DynObjCatElement;

// ---------------------------------------------------------------------------

class DynObjCatalog
{
            // nicht definiert:
                        DynObjCatalog   (const DynObjCatalog&);
        DynObjCatalog&  operator=       (const DynObjCatalog&);
    public:
                 DynObjCatalog ();
                ~DynObjCatalog ();
                     /* loescht alle DynamicalObject, deren DynObjCatElement
                        sich noch im Katalog befinden. */

            void    InsertDynamicalObject (DynamicalObject* s);
                    /* verknuepft s mit mit einem neuen DynObjCatElement,
                       dass in den Katalog eingefuegt wird. */
    static  void    RemoveDynamicalObject (DynamicalObject* s);
                    /* entfernt das DynObjCatElement von s aus dem Katalog.*/
            void    removeDynamicalObject (DynamicalObject* s);
                    /* entfernt das DynObjCatElement von s aus dem Katalog.*/
    private:
        DynObjCatElement*   first;
        DynObjCatElement*   last;
};

// ---------------------------------------------------------------------------

#endif  // DYNOBJCATALOG_H
```

dyobjcat.cc

```
// --------------------------------------------------------------------------
//
// Datei
//          dyobjcat.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// --------------------------------------------------------------------------

#include "dyobjcat.h"

#include "dyobject.h"

#include <assert.h>

// --------------------------------------------------------------------------

class DynObjCatElement
{
    /*  darf nur vom DynObjCatalog behandelt werden. Dies
        schliesst konstruktion und Destruktion ein.
    */
        friend class DynObjCatalog;
    private:
            DynObjCatElement (  DynamicalObject*    DynObj,
                                DynObjCatElement*   pred,
                                DynObjCatalog*      dynObjCatalog)
                            // wird immer nur am Ende eingefuegt
                            :   prev        (pred),
                                next        (0),
                                dynObj      (DynObj),
                                dynObjCat   (dynObjCatalog)
                            {
                                if (pred)
                                    pred->next = this;
                            }
            ~DynObjCatElement ()
                            {   // nur die Luecke schliessen
                                if (prev)
                                    prev->next = next;
                                if (next)
                                    next->prev = prev;
                                dynObj      = 0;
                                dynObjCat   = 0;
                            }

        DynObjCatElement*   prev;
        DynObjCatElement*   next;
        DynamicalObject*    dynObj;
        DynObjCatalog*      dynObjCat;  // nur zur Sicherheit
};

// --------------------------------------------------------------------------
// --------------------------------------------------------------------------

DynObjCatalog::DynObjCatalog ()
    :   first   (0),
        last    (0)
    {}

// --------------------------------------------------------------------------

DynObjCatalog::~DynObjCatalog ()
    {
        DynObjCatElement* doce = first;
        while (doce)
        {   //  erst das Garbage-Flag setzen
            doce->dynObj->isGarbage = true;
            doce = doce->next;
        }
```

```
        // dann loeschen
        while (first)
            delete first->dynObj;   // ruft RemoveDynamicalObject auf
    }

// ----------------------------------------------------------------------------

void DynObjCatalog::InsertDynamicalObject (DynamicalObject* d)
    {
        d->catElement = last = new DynObjCatElement (d, last, this);
        if (!first)
            first = last;
    }

// ----------------------------------------------------------------------------
// Klassen-Methode

void DynObjCatalog::RemoveDynamicalObject (DynamicalObject* d)
    {
        d->catElement->dynObjCat->removeDynamicalObject (d);
    }

// ----------------------------------------------------------------------------

void DynObjCatalog::removeDynamicalObject (DynamicalObject* d)
    {
        DynObjCatElement* doce = d->catElement;
        if (doce == first)
            first = doce->next;
        if (doce == last)
            last = doce->prev;
        delete doce;
    }

// ----------------------------------------------------------------------------
```

dyobject.h

```
// ----------------------------------------------------------------------------
//
// Datei
//          dyobject.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ----------------------------------------------------------------------------

#ifndef DYNAMICALOBJECT_H
#define DYNAMICALOBJECT_H

// ----------------------------------------------------------------------------

#include "modelcom.h"   // Basisklasse

#include "boolean.h"
#include "str.h"

// ----------------------------------------------------------------------------

class Model;
class DynObjCatElement;

// ----------------------------------------------------------------------------

class DynamicalObject : public ModelComponent
{
        friend class DynObjCatalog;
```

```
        DynamicalObject&    operator= (const DynamicalObject&);
            //  nicht definiert
    public:
                    DynamicalObject (   Model&  owner,
                                const   String& name = "",
                                        bool    showInTrace = true);
                    DynamicalObject (const DynamicalObject&);
        virtual    ~DynamicalObject ();

                void    DeleteOnTermination ();

                String  ClassName () const;

                bool    CheckDeleteOnTermination() const;
                bool    IsGarbage() const;
                            // wird nur intern benoetigt
    private:
        bool                delOnTermination;
        bool                isGarbage;  // wird von ~DynObjCatalog gesetzt
        DynObjCatElement*   catElement; // Verweis fuer effiezientes loeschen
};

// ---------------------------------------------------------------------------

#endif  // DYNAMICALOBJECT_H
```

dyobject.cc

```
// ---------------------------------------------------------------------------
//
// Datei
//          dyobject.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#include "dyobject.h"

#include "experimm.h"

#include <iostream.h>   // nur zum Testen

// ---------------------------------------------------------------------------

static const char* className = "DynamicalObject";

// ---------------------------------------------------------------------------

DynamicalObject::DynamicalObject (Model& owner, const String& name,
                                  bool showInTrace)
    :   ModelComponent      (owner, name, showInTrace),
        delOnTermination    (false),
        isGarbage           (false),
        catElement          (0)
    {
        ExperimentManager::Instance().Register (this);
            // meldet mich beim DynObjCatalog an, der catElement setzt
    }

// ---------------------------------------------------------------------------

DynamicalObject::DynamicalObject (const DynamicalObject& rhs)
    :   ModelComponent      (rhs),
        delOnTermination    (rhs.delOnTermination),
        isGarbage           (false),
        catElement          (0)
    {
```

```
        ExperimentManager::Instance().Register (this);
            // meldet mich beim DynObjCatalog an, der catElement setzt
    }

// ---------------------------------------------------------------------------

DynamicalObject::~DynamicalObject ()
    {
        ExperimentManager::Instance().DeRegister (this);
    }

// ---------------------------------------------------------------------------

void DynamicalObject::DeleteOnTermination ()
    {
        delOnTermination = true;
    }

// ---------------------------------------------------------------------------

String DynamicalObject::ClassName () const
    {
        return className;
    }

// ---------------------------------------------------------------------------

bool DynamicalObject::CheckDeleteOnTermination () const
    {
        return delOnTermination;
    }

// ---------------------------------------------------------------------------

bool DynamicalObject::IsGarbage () const
    {
        return isGarbage;
    }

// ---------------------------------------------------------------------------
```

eacc.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          eacc.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#ifndef EXPERIMENTACCESSORY_H
#define EXPERIMENTACCESSORY_H

// ---------------------------------------------------------------------------

#include "distman.h"
#include "messagem.h"
#include "outputm.h"
#include "resdb.h"
#include "simclock.h"
#include "schedule.h"

// ---------------------------------------------------------------------------
// alles, was zur Verwaltung eines Experimentes notwendig ist

class ExperimentAccessory
```

```
{
        friend class ExperimentManager;
        friend class Experiment;

    public:
                        ExperimentAccessory (Experiment&);
                        ~ExperimentAccessory ();

    private:
        Scheduler           scheduler;
        SimClock&           simClock;
        MessageManager      messageManager;
        OutputManager       errorManager,
                            traceManager,
                            debugManager,
                            reportManager;
        DistribManager      distribManager;
        ResourceDB          resourceDB;
};

// ---------------------------------------------------------------------------

#endif  // EXPERIMENTACCESSORY_H
```

eacc.cc

```
// ---------------------------------------------------------------------------
//
// Datei
//          eacc.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#include "eacc.h"

#include "experime.h"
#include "expopts.h"
#include "messagem.h"
#include "outputm.h"
#include "resdb.h"
#include "schedule.h"
#include "simclock.h"

// ---------------------------------------------------------------------------

ExperimentAccessory::ExperimentAccessory (Experiment& e)
    :   scheduler           (e.GetOpts().Epsilon()),
        simClock            (scheduler.GetSimClock()),
        messageManager      (),
        errorManager        (messageManager, e, Message::error),
        traceManager        (messageManager, e, Message::trace),
        debugManager        (messageManager, e, Message::debug),
        reportManager       (messageManager, e, Message::report),
        distribManager      (),
        resourceDB          ()
    {}

// ---------------------------------------------------------------------------

ExperimentAccessory::~ExperimentAccessory ()
    {}

// ---------------------------------------------------------------------------
```

emessage.h

```
// --------------------------------------------------------------------------
//
// Datei
//          emessage.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// --------------------------------------------------------------------------

#ifndef EMESSAGE_H
#define EMESSAGE_H

// --------------------------------------------------------------------------

#include "msgtypes.h"
#include "str.h"

// --------------------------------------------------------------------------

class CustomErrorMessage : public LocatableMessage
{
    public:
            CustomErrorMessage (const String& where, const String& what,
                                const String& conseq, const String& hint,
                                CodeType ct = Message::normalError);

        virtual String      Description () const;
    private:
        const   String  errorText;
};

// --------------------------------------------------------------------------
// Globale Fehlermeldungen

class CustomGlobalErrorMessage : public GlobalErrorMessage
{
    public:
        CustomGlobalErrorMessage (const String& where, const String& what,
                                  const String& conseq, const String& hint,
                                  CodeType ct = Message::normalError);

        virtual String      Description () const;
    private:
        const   String  errorText;
};

// --------------------------------------------------------------------------

#endif
```

emessage.cc

```
// --------------------------------------------------------------------------
//
// Datei
//          emessage.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
```

```
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#include "emessage.h"
#include "entity.h"
#include "msgcomp.h"

#include "str.h"

// ---------------------------------------------------------------------------
// Klasse: CustomErrorMessage
// ---------------------------------------------------------------------------

CustomErrorMessage::CustomErrorMessage (const String& where,
                                        const String& what,
                                        const String& conseq,
                                        const String& Hint,
                                        CodeType      ct)
    :   LocatableMessage (error, ct, where, new MsgCnsqCustom (conseq),
                                            new MsgHintCustom (Hint)),
        errorText(what)
    {}

// ---------------------------------------------------------------------------

String CustomErrorMessage::Description () const
    {   return errorText; }

// ---------------------------------------------------------------------------
// Klasse: CustomGlobalErrorMessage
// ---------------------------------------------------------------------------

CustomGlobalErrorMessage::CustomGlobalErrorMessage (const String& where,
                                                    const String& what,
                                                    const String& conseq,
                                                    const String& Hint,
                                                    CodeType      ct)
    :   GlobalErrorMessage (where, ct, new MsgCnsqCustom (conseq),
                                       new MsgHintCustom (Hint)),
        errorText(what)
    {}

// ---------------------------------------------------------------------------

String CustomGlobalErrorMessage::Description () const
    {   return errorText; }

// ---------------------------------------------------------------------------
```

entity.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          entity.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#ifndef ENTITY_H
#define ENTITY_H

// ---------------------------------------------------------------------------
```

```
#include "schedula.h"   // Basisklasse

#include "boolean.h"
#include "nobject.h"
#include "schedula.h"
#include "simtime.h"
#include "str.h"

// ---------------------------------------------------------------------------

typedef int PriorityT;

// ---------------------------------------------------------------------------

class Event;

class Entity : public Schedulable
    /*  Die Klasse Entity dient als Basisklasse fuer alle dynamischen Objekte,
        die von Ereignissen assoziiert werden koennen.
        Entities sind prinzipiell passive Objekte, da sie kein
        eigenes Verhalten besitzen wie Prozesse oder Ereignis-Routinen.
    */
{
        friend class Process;   // setzt isProcess
        friend class QueueLink; // verwaltet Warteschlangenzugehoerigkeiten

                    Entity (const Entity&);     // nicht definiert!
                    operator= (const Entity&);  // nicht definiert!
    public:
                    Entity (Model& owner, const String& name = "",
                            bool showInTrace = true);
        virtual    ~Entity ();

                void        Delete ();
                    /*  Das aktuelle (current) Entity darf
                        nicht geloescht werden. Delete sorgt
                        dafuer, dass das Entity bei der naechsten
                        Gelegenheit vom System geloescht wird */

                bool        IsNullEntity () const;
                                // Handelt es sich um das Pseudo-Entity?
                bool        IsProcess () const;
                                // Handelt es sich um einen Prozess?

                void        Schedule (SimTime dt, Event& ev);
                    /*  Vormerken des Entities mit dem Ereignis ev zum
                        Zeitpunkt now + dt. */

                void        ScheduleBefore (Schedulable& before, Event& ev);

                void        ScheduleAfter (Schedulable& after, Event& ev);

                PriorityT   GetPriority () const;
                    /*  Abfragen der Prioritaet des Entities
                        je groesser die Prioritaet, desto mehr wird das
                        Entity beim Einreihen in Warteschlangen bevorzugt */

                PriorityT   SetPriority (const PriorityT newPriority);
                    /*  Setzen der Prioritaet. Sie hat Einfluss auf die
                        Rangfolge in allen im- und explizieten Warteschlangen.
                        Die neue Prioritaet wird zurueckgegeben. */

                QueueOption GetQueueOption() const;
                QueueOption SetQueueOption(QueueOption newOption);

                bool        operator== (const Entity& en) const;
                                // Ist das Entity mit en identitsch?
                bool        operator!= (const Entity& en) const;
                                // Ist das Entity mit en nicht identitsch?
                bool        operator<  (const Entity& en) const;
                                // fuer Warteschlangen-Sortierung
                bool        operator>  (const Entity& en) const;
                                // fuer Warteschlangen-Sortierung
                bool        operator<= (const Entity& en) const;
                                // fuer Warteschlangen-Sortierung
                bool        operator>= (const Entity& en) const;
                                // fuer Warteschlangen-Sortierung
                String      ClassName () const;

    private:
        bool        isProcess;
        PriorityT   priority;
        QueueLink*  qlink;
```

```
        QueueOption queueOption;
};

// ---------------------------------------------------------------------------

#endif  // ENTITY_H
```

entity.cc

```
// ---------------------------------------------------------------------------
//
// Datei
//          entity.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#include "entity.h"

#include "boolean.h"
#include "experimm.h"
#include "event.h"
#include "model.h"
#include "process.h"
#include "qlink.h"
#include "qimpl.h"
#include "schedule.h"
#include "simtime.h"
#include "msgsched.h"

#include <assert.h>

// ---------------------------------------------------------------------------

static const char* className = "Entity";

// ---------------------------------------------------------------------------

Entity::Entity (Model& owner, const String& name, bool trace)
    :   Schedulable (owner, name, trace),
        isProcess   (false),
        priority    (0),
        qlink       (0),
        queueOption (owner.GetQueueOption())
    {}

// ---------------------------------------------------------------------------

Entity::~Entity ()
    {
        const char* where = "Entity::~Entity";
        while (qlink)
            qlink->GetQueue().Remove(*this);

        if (this == &CurrentEntity() && !IsNullEntity())
            Error ( "deletion of the current entity " + QuotedName(),
                    where, "experiment is aborted"
                    "use 'Delete' to delete an entity");
    }

// ---------------------------------------------------------------------------

void Entity::Delete ()
    {
        const char* where = "Entity::Delete";

        if (!valid (className, where))
```

```
            return;

        if (IsNullEntity())
        {
            Warning (   "the NullEntity must not be deleted",
                        where, "ignored");
            return;
        }
        if (IsScheduled())
        {
            Warning (   "deletion of the scheduled entity " + QuotedName(),
                        where, "ignored");
            return;
        }

        // aus Warteschlangen entfernen
        if (qlink)
        {
            Warning (   "deletion of the entity " + QuotedName() +
                        "which waits in a queue", where,
                        QuotedName() + " will be removed from the queue(s)");
            while (qlink)
                qlink->GetQueue().Remove(*this);
        }

        if (TraceIsOn())
            SendMessage (TrcDelete (*this));

        ExperimentManager::Instance().GetScheduler(*this).Terminate(*this);
    }

// ---------------------------------------------------------------------------

void Entity::Schedule (SimTime dt, Event& ev)
    {
        const char* where = "Entity::Schedule";

        // this pruefen
        if (!valid (className, where))
            return;
        if (IsScheduled())
        {
            Warning (   "attempt to schedule an already scheduled"
                      + ClassName(),
                        where, "ignored");
            return;
        }

        // Event pruefen
        if (!valid (ev, "Event", where))
            return;
        if (!IsExperimentCompatible (ev))
        {
            Warning ("attemp to mix components of different experiments",
                     where, "ignored");
            return;
        }
        if (!IsModelCompatible (ev))
        {
            Warning (   "attempt to schedule an entity with an "
                        "incompatible event",
                        where, "ignored");
            return;
        }
        if (ev.IsScheduled())
        {
            Warning (   "attempt to schedule an already scheduled "
                      + ev.ClassName(),
                        where, "ignored");
            return;
        }

        if (dt < 0.0 && dt != NOW())
        {
            Warning (   "negativ dt [" + String(dt.Time()) + ']',
                        where, "0.0 is used");
            dt = 0.0;
        }

        // OK
        if (TraceIsOn())
            SendMessage (TrcSchedule (dt, ev, *this));

        Scheduler& scheduler = ExperimentManager::Instance().
```

```
                               GetScheduler(*this);
        scheduler.Schedule (dt, ev, *this);
    }


// ---------------------------------------------------------------------------

void Entity::ScheduleBefore (Schedulable& before, Event& ev)
    {
        const char* where = "Entity::ScheduleBefore";

        // this pruefen
        if (!valid (className, where))
            return;
        if (IsScheduled())
        {
            Warning (   "attempt to schedule an already scheduled "
                      + ClassName(),
                        where, "ignored");
            return;
        }

        // Event pruefen
        if (!valid (ev, "Event", where))
            return;
        if (!IsExperimentCompatible (ev))
        {
            Warning ("attemp to mix components of different experiments",
                     where, "ignored");
            return;
        }
        if (!IsModelCompatible (ev))
        {
            Warning (   "attempt to schedule an entity with an "
                        "incompatible event",
                        where, "ignored");
            return;
        }
        if (ev.IsScheduled())
        {
            Warning (   "attempt to schedule an already scheduled "
                      + ev.ClassName(),
                        where, "ignored");
            return;
        }

        // before pruefen
        if (!valid (before, "Schedulable", where))
            return;
        if (!IsExperimentCompatible (before))
        {
            Warning (   "attempt to schedule an entity before an object "
                        "of another experiment",
                        where, "ignored");
            return;
        }
        if (before.IsCurrent() && CurrentProcess().IsNullProcess())
        {
            Warning (   "attempt to schedule an entity before the "
                        "current event",
                        where, "ignored");
            return;
        }

        if (!before.IsScheduled() && !before.IsCurrent())
        {
            Warning (   "attempt to schedule an entity before an "
                        "unscheduled object",
                        where, "ignored");
            return;
        }


        // OK
        if (TraceIsOn())
            SendMessage (TrcScheduleBefore (before, ev, *this));

        Scheduler& scheduler = ExperimentManager::Instance().
                               GetScheduler(*this);
        scheduler.ScheduleBefore (before, ev, *this);
    }

// ---------------------------------------------------------------------------
```

```
void Entity::ScheduleAfter (Schedulable& after, Event& ev)
    {
        const char* where = "Entity::ScheduleAfter";

        // this pruefen
        if (!valid (className, where))
            return;
        if (IsScheduled())
        {
            Warning (   "attempt to schedule an already scheduled"
                      + ClassName(),
                        where, "ignored");
            return;
        }

        // Event pruefen
        if (!valid (ev, "Event", where))
            return;
        if (!IsExperimentCompatible (ev))
        {
            Warning ("attemp to mix components of different experiments",
                     where, "ignored");
            return;
        }
        if (!IsModelCompatible (ev))
        {
            Warning (   "attempt to schedule an entity with an "
                        "incompatible event",
                        where, "ignored");
            return;
        }
        if (ev.IsScheduled())
        {
            Warning (   "attempt to schedule an already scheduled"
                      + ev.ClassName(),
                        where, "ignored");
            return;
        }

        // after pruefen
        if (!valid (after, "Schedulable", where))
            return;
        if (!IsExperimentCompatible (after))
        {
            Warning (   "attempt to schedule an entity after an object of "
                        "another experiment",
                        where, "ignored");
            return;
        }
        if (!after.IsScheduled() && !after.IsCurrent())
        {
            Warning (   "attempt to schedule an entity after an "
                        "unscheduled object",
                        where, "ignored");
            return;
        }

        // OK
        if (TraceIsOn())
            SendMessage (TrcScheduleAfter (after, ev, *this));

        Scheduler& scheduler = ExperimentManager::Instance().
                                   GetScheduler(*this);
        scheduler.ScheduleAfter (after, ev, *this);
    }

// ---------------------------------------------------------------------------

PriorityT Entity::GetPriority () const
    {
        const char* where = "Entity::GetPriority";

        if (!valid (className, where))
            return 0;

        return priority;
    }

// ---------------------------------------------------------------------------

PriorityT Entity::SetPriority (const PriorityT newPriority)
    {
        const char* where = "Entity::SetPriority";
```

```
        if (!valid (className, where))
            return 0;

        return priority = newPriority;
    }

// ------------------------------------------------------------------------

bool Entity::IsNullEntity () const
    {
        return this == &NullEntity();
    }

// ------------------------------------------------------------------------

bool Entity::IsProcess () const
    {
        return isProcess;
    }

// ------------------------------------------------------------------------

String Entity::ClassName () const
    {
        return className;
    }

// ------------------------------------------------------------------------

QueueOption Entity::GetQueueOption () const
    {
        return queueOption;
    }

// ------------------------------------------------------------------------

QueueOption Entity::SetQueueOption (QueueOption newOption)
    {
        if (queueOption == newOption)
            return queueOption;

        QueueOption temp = queueOption;

        // evtl. pruefen, ob bereits in mehreren Queues

        queueOption = newOption;
        return temp;
    }

// ------------------------------------------------------------------------

bool Entity::operator==(const Entity& en) const
    {
        return this == &en;
    }

// ------------------------------------------------------------------------

bool Entity::operator!=(const Entity& en) const
    {
        return this != &en;
    }

// ------------------------------------------------------------------------

bool Entity::operator< (const Entity& en) const
    {
        return priority < en.priority;
    }

// ------------------------------------------------------------------------

bool Entity::operator> (const Entity& en) const
    {
        return priority > en.priority;
    }

// ------------------------------------------------------------------------

bool Entity::operator<= (const Entity& en) const
    {
        return priority <= en.priority;
    }
```

```
// --------------------------------------------------------------------------

bool Entity::operator>= (const Entity& en) const
    {
        return priority >= en.priority;
    }

// --------------------------------------------------------------------------
```

event.h

```
// --------------------------------------------------------------------------
//
// Datei
//          event.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// --------------------------------------------------------------------------

#ifndef EVENT_H
#define EVENT_H

// --------------------------------------------------------------------------

#include "schedula.h"   // Oberklasse

#include "entity.h"
#include "simtime.h"
#include "str.h"
#include "boolean.h"

// --------------------------------------------------------------------------

class Event : public Schedulable
    /*  Event dient als Basisklasse fuer alle Simulations-Ereignisse. Dabei
        muss EventRoutine in der Unterklasse definiert werden um die
        Reaktionen auf das Eintreten des Ereignisses zu beschreiben.
    */
{
        friend class Scheduler;     // Zugriff auf EventRoutine
        friend class ExternalEvent; // Zugriff auf isExternal
    public:
                    Event (     Model&  owner,
                        const   String& name = "",
                                bool    showInTrace = true);
        virtual    ~Event ();

                void    Schedule (SimTime dt, Entity& en);
                    /*  Vormerken des Ereignisses zum Zeitpunkt now + dt.
                        en2 ist das mit Eintreten dieses Ereignisses
                        assozierte Entity. */

                void    ScheduleBefore (Schedulable& before, Entity& en);

                void    ScheduleAfter (Schedulable& after, Entity& en);

                bool    IsNullEvent () const;
                bool    IsExternal  () const;

                String  ClassName () const;

    protected:
        virtual void    EventRoutine (Entity& entity) = 0;
                    /*  Hier muss in den Unterklassen definiert werden, wie
                        auf das Eintreten dieses Ereignisses reagiert werden
                        soll. */
```

```
    private:
        bool    isExternal;
};

// ----------------------------------------------------------------------------

class ExternalEvent : public Event
    /*  Event dient als Basisklasse fuer alle externe Ereignisse. Dabei muss
        ExternalEventRoutine in der Unterklasse definiert werden
        Reaktionen um die auf das Eintreten des Ereignisses zu beschreiben.
    */
{
    public:
                        ExternalEvent (     Model&  owner,
                                    const   String& name = "",
                                            bool    showInTrace = true);

                void    Schedule (SimTime dt);
                    /*  Vormerken des Ereignisses zum Zeitpunkt now + dt. */

                void    ScheduleBefore (Schedulable& before);

                void    ScheduleAfter (Schedulable& after);

                String  ClassName () const;

    protected:
        virtual void    ExternalEventRoutine () = 0;
                    /*  Hier muss in den Unterklassen definiert werden, wie
                        auf das Eintreten dieses Ereignisses reagiert werden
                        soll. */

                void    EventRoutine (Entity&);
                    /*  darf nicht benutzt werden, produziert eine Warnung
                        und ruft anschliessend ExternalEventRoutine auf. */
};

// ----------------------------------------------------------------------------

#endif
```

event.cc

```
// ----------------------------------------------------------------------------
//
// Datei
//          event.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ----------------------------------------------------------------------------

#include "event.h"

#include "entity.h"
#include "experimm.h"
#include "schedule.h"
#include "msgsched.h"

#include <assert.h>

// ----------------------------------------------------------------------------

static const char* className = "Event";

// ----------------------------------------------------------------------------

Event::Event (Model& owner, const String& name, bool showInTrace)
```

```
    :   Schedulable (owner, name, showInTrace),
        isExternal  (false)
    {
    }

// -----------------------------------------------------------------------------

Event::~Event ()
    {
        const char* where = "Event::~Event()";

        if (this == &CurrentEvent() && !IsNullEvent())
            Error ( "deletion of the current event " + QuotedName(),
                    where);
    }

// -----------------------------------------------------------------------------

void Event::Schedule (SimTime dt, Entity& en)
    {
        const char* where = "Event::Schedule";

        // this pruefen
        if (!valid (className, where))
            return;
        if (IsScheduled())
        {
            Warning (   "attempt to schedule an already scheduled"
                      + ClassName(),
                        where, "ignored");
            return;
        }

        if (!valid (en, "Entity", where))
            return;
        if (!IsExperimentCompatible (en))
        {
            Warning ("attemp to mix components of different experiments",
                     where, "ignored");
            return;
        }
        if (!IsModelCompatible (en))
        {
            Warning (   "attempt to schedule an event with an "
                        "incompatible entity",
                        where, "ignored");
            return;
        }
        if (en.IsScheduled())
        {
            Warning (   "attempt to schedule an already scheduled"
                      + en.ClassName(),
                        where, "ignored");
            return;
        }

        if (dt < 0.0 && dt != NOW())
        {
            Warning (   "negativ dt [" + String(dt.Time()) + ']',
                        where, "0.0 is used");
            dt = 0.0;
        }

        // OK
        if (TraceIsOn())
            SendMessage (TrcSchedule(dt, *this, en));

        Scheduler& scheduler = ExperimentManager::Instance().
                                   GetScheduler(*this);
        scheduler.Schedule (dt, *this, en);
    }

// -----------------------------------------------------------------------------

void Event::ScheduleBefore (Schedulable& before, Entity& en)
    {
        const char* where = "Event::ScheduleBefore";

        // this pruefen
        if (!valid (className, where))
            return;
        if (IsScheduled())
        {
            Warning (   "attempt to schedule an already scheduled"
```

```
                      + ClassName(),
                        where, "ignored");
            return;
        }

        if (!valid (en, "Entity", where))
            return;
        if (!IsExperimentCompatible (en))
        {
            Warning ("attemp to mix components of different experiments",
                     where, "ignored");
            return;
        }
        if (!IsModelCompatible (en))
        {
            Warning (   "attempt to schedule an event with an "
                        "incompatible entity",
                        where, "ignored");
            return;
        }
        if (en.IsScheduled())
        {
            Warning (   "attempt to schedule an already scheduled"
                      + en.ClassName(),
                        where, "ignored");
            return;
        }

        // before pruefen
        if (!valid (before, "Schedulable", where))
            return;
        if (!IsExperimentCompatible (before))
        {
            Warning (   "attempt to schedule an event before an object of "
                        "another experiment",
                        where, "ignored");
            return;
        }
        if (before.IsCurrent() && CurrentProcess().IsNullProcess())
        {
            Warning (   "attempt to schedule an event before the "
                        "current event",
                        where, "ignored");
            return;
        }

        if (!before.IsScheduled() && !before.IsCurrent())
        {
            Warning (   "attempt to schedule an event before an "
                        "unscheduled object",
                        where, "ignored");
            return;
        }

        // OK
        if (TraceIsOn())
            SendMessage (TrcScheduleBefore(before, *this, en));

        Scheduler& scheduler = ExperimentManager::Instance().
                                   GetScheduler(*this);
        scheduler.ScheduleBefore (before, *this, en);
    }

// ----------------------------------------------------------------------------

void Event::ScheduleAfter (Schedulable& after, Entity& en)
    {
        const char* where = "Event::ScheduleAfter";

        // this pruefen
        if (!valid (className, where))
            return;
        if (IsScheduled())
        {
            Warning (   "attempt to schedule an already scheduled"
                      + ClassName(),
                        where, "ignored");
            return;
        }

        if (!valid (en, "Entity", where))
            return;
        if (!IsExperimentCompatible (en))
        {
```

```
            Warning ("attemp to mix components of different experiments",
                     where, "ignored");
            return;
        }
        if (!IsModelCompatible (en))
        {
            Warning (   "attempt to schedule an event with an "
                        "incompatible entity",
                        where, "ignored");
            return;
        }
        if (en.IsScheduled())
        {
            Warning (   "attempt to schedule an already scheduled"
                        + en.ClassName(),
                        where, "ignored");
            return;
        }

        // after pruefen
        if (!valid (after, "Schedulable", where))
            return;
        if (!IsExperimentCompatible (after))
        {
            Warning (   "attempt to schedule an event after an object "
                        "of another experiment",
                        where, "ignored");
            return;
        }
        if (!after.IsScheduled() && !after.IsCurrent())
        {
            Warning (   "attempt to schedule an entity after an "
                        "unscheduled object",
                        where, "ignored");
            return;
        }

        // OK
        if (TraceIsOn())
            SendMessage (TrcScheduleAfter(after, *this, en));

        Scheduler& scheduler = ExperimentManager::Instance().
                                   GetScheduler(*this);
        scheduler.ScheduleBefore (after, *this, en);
    }

// ---------------------------------------------------------------------------

String Event::ClassName () const
    {
        return className;
    }

// ---------------------------------------------------------------------------

bool Event::IsNullEvent () const
    {
        return this == &NullEvent();
    }

// ---------------------------------------------------------------------------

bool Event::IsExternal () const
    {
        return isExternal;
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

static const char* className2 = "ExternalEvent";

// ---------------------------------------------------------------------------

ExternalEvent::ExternalEvent (Model& owner, const String& name,
                              bool showInTrace)
    :   Event(owner, name, showInTrace)
    {
        isExternal = true;
    }

// ---------------------------------------------------------------------------

void ExternalEvent::EventRoutine (Entity& en)
```

```
    {
        const char* where = "ExternalEvent::EventRoutine";

        if (!en.IsNullEntity())
            Warning ("EventRoutine of the ExternalEvent " + QuotedName() +
                     " was called with entity " + en.QuotedName(),
                     where, "ExternalEventRoutine is called instead");

        ExternalEventRoutine ();
    }

// --------------------------------------------------------------------------

void ExternalEvent::Schedule (SimTime dt)
    {
        const char* where = "ExternalEvent::Schedule";

        // this pruefen
        if (!valid (className2, where))
            return;
        if (IsScheduled())
        {
            Warning (   "attempt to schedule an already scheduled"
                      + ClassName(),
                        where, "ignored");
            return;
        }

        if (dt < 0.0 && dt != NOW())
        {
            Warning (   "negativ dt [" + String(dt.Time()) + ']',
                        where, "0.0 is used");
            dt = 0.0;
        }

        // OK
        if (TraceIsOn())
            SendMessage (TrcSchedule(dt, *this, NullEntity()));

        Scheduler& scheduler = ExperimentManager::Instance().
                                   GetScheduler (*this);
        scheduler.Schedule (dt, *this, NullEntity());
    }

// --------------------------------------------------------------------------

void ExternalEvent::ScheduleBefore (Schedulable& before)
    {
        const char* where = "ExternalEvent::ScheduleBefore";

        // this pruefen
        if (!valid (className2, where))
            return;
        if (IsScheduled())
        {
            Warning (   "attempt to schedule an already scheduled"
                      + ClassName(),
                        where, "ignored");
            return;
        }

        // before pruefen
        if (!valid (before, "Schedulable", where))
            return;
        if (!IsExperimentCompatible (before))
        {
            Warning (   "attempt to schedule an event before an "
                        "object of another experiment",
                        where, "ignored");
            return;
        }
        if (before.IsCurrent() && CurrentProcess().IsNullProcess())
        {
            Warning (   "attempt to schedule an event before the "
                        "current event",
                        where, "ignored");
            return;
        }
        if (!before.IsScheduled() && !before.IsCurrent())
        {
            Warning (   "attempt to schedule an event before an "
                        "unscheduled object",
                        where, "ignored");
            return;
```

```
        }

        // OK
        if (TraceIsOn())
            SendMessage (TrcScheduleBefore(before, *this, NullEntity()));

        Scheduler& scheduler = ExperimentManager::Instance().
                                    GetScheduler(*this);
        scheduler.ScheduleBefore (before, *this, NullEntity());
    }

// ---------------------------------------------------------------------------

void ExternalEvent::ScheduleAfter (Schedulable& after)
    {
        const char* where = "ExternalEvent::ScheduleAfter";

        // this pruefen
        if (!valid (className2, where))
            return;
        if (IsScheduled())
        {
            Warning (   "attempt to schedule an already scheduled"
                      + ClassName(),
                        where, "ignored");
            return;
        }

        // after pruefen
        if (!valid (after, "Schedulable", where))
            return;
        if (!IsExperimentCompatible (after))
        {
            Warning (   "attempt to schedule an event after an object of "
                        "another experiment",
                        where, "ignored");
            return;
        }
        if (!after.IsScheduled() && !after.IsCurrent())
        {
            Warning (   "attempt to schedule an entity after an "
                        "unscheduled object",
                        where, "ignored");
            return;
        }

        // OK
        if (TraceIsOn())
            SendMessage (TrcScheduleAfter(after, *this, NullEntity()));

        Scheduler& scheduler = ExperimentManager::Instance().
                                    GetScheduler(*this);
        scheduler.ScheduleBefore (after, *this, NullEntity());
    }

// ---------------------------------------------------------------------------

String ExternalEvent::ClassName () const
    {
        return className2;
    }

// ---------------------------------------------------------------------------
```

eventlis.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          eventlis.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
```

```
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------
//
// Beschreibung
//
//          Abstrakte Ereiginisliste: Sowohl EventList als auch
//          EventNote muessen konkretiesert werden und bilden
//          gemeinsam die Implementation eines konkreten Algorithmus
//          zur Ereignis-Notiz-Verwaltung Fuer EventList muss nur
//          die Fabrikmethode NewEventNote() ueberschrieben werden,
//          die dann eine zur Implementation passende Notiz erzeugt
//          und liefert. Alle anderen Methoden koennen auf
//          abstrakter Ebene definiert werden, und muessen in der
//          konkreten Version nicht redefiniert werden.
//
// ---------------------------------------------------------------------------

#ifndef EVENTLIST_H
#define EVENTLIST_H

// ---------------------------------------------------------------------------

#include "boolean.h"
#include "simtime.h"
#include "entity.h"
#include "event.h"

// ---------------------------------------------------------------------------

class Model;

// ---------------------------------------------------------------------------

class EventNote
{
        friend class Scheduler;

    public:
        EventNote (const    SimTime& t = 0.0,
                            Event* ev = &ModelComponent::NullEvent(),
                            Entity* en = &ModelComponent::NullEntity());
        EventNote (const    EventNote& eventNote);
            /*  kopiert die Verweise auf die Inhalte mit, aendert jedoch nicht
                die Verweise von den Inhalten auf die Ereignisnotiz!
            */
        virtual    ~EventNote ();

        // Vergleiche fuer die Reihenfolge in der Ereignisliste
        bool    operator==  (const EventNote& note) const;
        bool    operator!=  (const EventNote& note) const;
        bool    operator<   (const EventNote& note) const;
        bool    operator>   (const EventNote& note) const;
        bool    operator<=  (const EventNote& note) const;
        bool    operator>=  (const EventNote& note) const;

        virtual bool        IsScheduled () const = 0;  // noch in der Liste?

        SimTime             Time ();

    private:
        SimTime         time;
        Event*          event;
        Entity*         entity;
        Model*          model;
};

// ---------------------------------------------------------------------------

class EventList
    /*  Suchoperationen liefern 0, wenn nichts gefunden werden konnte.
        Bei Manipulatoren weist der Rueckgabewert 0 auf einen Fehler in
        der Benutzung der Erignisliste hin.
    */
{
    public:
        EventList ();
        EventList (EventList& source);  // in Unterklassen definieren!!!
            /*  Neue Liste wird mit neuen Ereignisnotizen angelegt, die
                denen in source entsprechen. Dabei erfolgt exakt die
                geliche Sortierung. Verweise von Schedulable auf
```

```
                EventNote bleiben unangetastet, so dass die alte Liste
                (source) gueltig bleibt.
            */
        virtual ~EventList ();

        virtual EventList*  Clone() const = 0;  // fuer das Prototyp-Muster
                                // erzeugt leere Liste gleichen Typs
        virtual EventList*  Clone (const EventList& source) const = 0;
                               // klont mit Hilfe des Copy-Konstruktors

        virtual EventNote*  FirstEventNote () const = 0;
        virtual EventNote*  LastEventNote () const = 0;

        virtual EventNote*  NewEventNote
                              (const SimTime& t = 0.0,
                               Event* event = &ModelComponent::NullEvent(),
                               Entity* entity = &ModelComponent::NullEntity()
                             ) const = 0;  // Fabrikmethode
        virtual EventNote*  NewEventNote (const EventNote& note) const = 0;
                                               // Fabrikmethode

        virtual EventNote*  NextEventNote (const EventNote* note) const = 0;
        virtual EventNote*  PrevEventNote (const EventNote* note) const = 0;

        virtual EventNote*  Insert          (EventNote* note) = 0;
        virtual EventNote*  InsertAsFirst   (EventNote* note) = 0;
        virtual EventNote*  InsertAsLast    (EventNote* note) = 0;
        virtual EventNote*  InsertBefore    (EventNote* before,
                                             EventNote* note) = 0;
        virtual EventNote*  InsertAfter     (EventNote* after,
                                             EventNote* note) = 0;

        virtual EventNote*  Remove (EventNote* note) = 0;
                              // entfernt, ohne zu loeschen
        virtual EventNote*  RemoveFirst () = 0;
                              // entfernt die erste Notiz

        virtual void        DeleteAll () = 0;
                              // loescht alle Notizen, nicht deren Inhalt
};

// ---------------------------------------------------------------------------

#endif  // EVENTLIST_H
```

eventlis.cc

```
// ---------------------------------------------------------------------------
//
// Datei
//          eventlis.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#include "eventlis.h"

#include "boolean.h"
#include "event.h"
#include "simtime.h"
#include <assert.h>

// ---------------------------------------------------------------------------
//class EventNote

EventNote::EventNote (const SimTime& t,
                      Event* ev,
```

```
                            Entity* en)
    :   time    (t),
        event   (ev),
        entity  (en),
        model   (0)
    {
        assert (ev);
        assert (en);
        if (!event->IsNullEvent())
            model = &event->GetModel();
        else
            model = &entity->GetModel();
    }

// ------------------------------------------------------------------------------
// Kopiersemantik

EventNote::EventNote (const EventNote& note)
    :   time    (note.time),
        event   (note.event),
        entity  (note.entity),
        model   (note.model)
    {}

// ------------------------------------------------------------------------------

EventNote::~EventNote ()
    {}

// ------------------------------------------------------------------------------
// fuer die Ordnung ist allein die Zeit entscheidend

bool EventNote::operator== (const EventNote& note) const
    {   return time == note.time; }

// ------------------------------------------------------------------------------

bool EventNote::operator!= (const EventNote& note) const
    {   return time != note.time; }

// ------------------------------------------------------------------------------

bool EventNote::operator< (const EventNote& note) const
    {   return time < note.time; }

// ------------------------------------------------------------------------------

bool EventNote::operator> (const EventNote& note) const
    {   return time > note.time; }

// ------------------------------------------------------------------------------

bool EventNote::operator<= (const EventNote& note) const
    {   return time <= note.time; }

// ------------------------------------------------------------------------------

bool EventNote::operator>= (const EventNote& note) const
    {   return time >= note.time; }

// ------------------------------------------------------------------------------

SimTime EventNote::Time ()
    {   return time; }

// ------------------------------------------------------------------------------
// class EventList
// ------------------------------------------------------------------------------

EventList::EventList ()
    {}

// ------------------------------------------------------------------------------

EventList::~EventList ()
    {}

// ------------------------------------------------------------------------------
```

experime.h

```
// -------------------------------------------------------------------------
//
// Datei
//          experime.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// -------------------------------------------------------------------------

#ifndef EXPERIMENT_H
#define EXPERIMENT_H

// -------------------------------------------------------------------------

class Model;
class ExperimentAccessory;
class Output;
class Message;

// -------------------------------------------------------------------------

#include "nobject.h"
#include "boolean.h"
#include "expopts.h"
#include "simtime.h"
#include "str.h"

// -------------------------------------------------------------------------

class Experiment : public NamedObject
{
        friend class ExperimentManager;

                        Experiment (const Experiment&);
                            /*  Copy-Konstruktor nicht implementiert! */
            Experiment& operator= (const Experiment&);
                            /*  Zuweisung nicht implementiert! */
    public:
                        Experiment (const String& name,
                                    const ExperimentOpts& = ExperimentOpts());

        virtual        ~Experiment ();

        virtual String  Description() const;
                            /*  soll eine Beschreibung des Experiments
                                liefern und wird im Report als erstes
                                ausgegeben. */
        virtual void    Rename (const String&) {};
                            //  ein Experiment darf nicht umbenannt werden

static  SimTime         NOW();

        void            Start (SimTime t = 0.0);
                            // Startet das Experiment mit Stratzeit t
        void            Continue ();
                            // Setzt ein unterbrochenes Experiment fort
        void            Stop (SimTime dt = SimTime::Now());
                            // Unterbricht ein laufendes Experiment in dt
        void            Report (SimTime dt = 0.0);
                            // Erzeugt einen Report in dt Zeiteinheiten
        void            Reset (SimTime dt = 0.0);
                            // Setzt die Statistik in dt Zeiteinheiten zurueck

        bool            TraceIsOn () const;
                            // ist der Trace eingeschaltet?
        void            TraceOn (SimTime dt = SimTime::Now());
                            // Schaltet den Trace ein
        void            TraceOff (SimTime dt = SimTime::Now());
                            // Schaltet den Trace aus
```

```
        bool            DebugIsOn () const;
                            // ist die Debug-Funktion eingeschaltet?
        void            DebugOn (SimTime dt = SimTime::Now());
                            // Schaltet die Debug-Funktion ein
        void            DebugOff (SimTime dt = SimTime::Now());
                            // Schaltet die Debug-Funktion aus

        void            SetSeed (long newSeed);
                            /* setzt den Startwert des Startwertgenerators
                               fuer die Zufallszahlenstroeme. Alle bereits
                               existierenden ZZ-Stroeme erhalten ebenfalls
                               einen neuen Startwert. */
        void            Antithetic ();
                            /* setzt alle Zufallszahlenstroeme des Experiments
                               auf antithetisch */

        enum            DeadlockLevelT {Off, Static, DynamicA, DynamicB};
        void            DeadlockLevel (DeadlockLevelT deadlockLevel);
                            /* Schaltet einen entsprechenden Level fuer die
                               Deadlockueberwachung ein. Nach dem Aufruf von
                               Start kann die Ueberwachung nur noch
                               ausgeschaltet werden. */
                            /* Vorgabe ist Static */

        Model&          GetModel();

        const
        ExperimentOpts& GetOpts() const;

        int             TimeWidth () const;        // Anzahl Stellen fuer
                                                   // Zeit-Ausgabe
        int             TimePrecision () const;    // Nachkommastellen + 1
        int             NameWidth () const;        // Namenlaenge inkl. Nummer
        int             NameNumberWidth () const;  // Anzahl der Stellen fuer
                                                   // die Namennummerierung
        SimTime         Epsilon() const;           // kleinste Zeiteinheit

        void            AddErrorOutput (Output&);
        void            AddDebugOutput (Output&);
        void            AddReportOutput(Output&);
        void            AddTraceOutput (Output&);

        void            RemoveErrorOutput (Output&);
        void            RemoveDebugOutput (Output&);
        void            RemoveReportOutput(Output&);
        void            RemoveTraceOutput (Output&);

        static  void        SetDesmoOut (ostream& out);
        static  void        SetDesmoErr (ostream& err);
        static  void        SetDesmoIn  (istream& in);
        static  void        ResetDesmoIO();
        static  ostream&    Out();
        static  ostream&    Err();
        static  istream&    In();
    protected:
                void    Warning (const String& what,
                                 const String& where,
                                 const String& Consequences = "",
                                 const String& hint         = "") const;
                void    Error   (const String& what,
                                 const String& where,
                                 const String& Consequences = "",
                                 const String& hint         = "") const;
                void    FatalError (const String& what,
                                    const String& where,
                                    const String& Consequences  = "",
                                    const String& hint          = "") const;
                void    SendMessage (const Message&) const;

    private:
        Model*                  model;
        enum {  start,
                running,
                stopped,
                aborted}        state;
        const ExperimentOpts    eOpts;
        ExperimentAccessory&    eAcc;
};

// -----------------------------------------------------------------------------

#endif  // EXPERIMENT_H
```

experime.cc

```
// --------------------------------------------------------------------------
//
// Datei
//          experime.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// --------------------------------------------------------------------------

#include "experime.h"

#include "experimm.h"
#include "eacc.h"
#include "distman.h"
#include "event.h"
#include "emessage.h"
#include "messagem.h"
#include "model.h"
#include "str.h"
#include "sysevent.h"

#include <assert.h>

// --------------------------------------------------------------------------

static const char* className = "Experiment";

// --------------------------------------------------------------------------

void Experiment::SetDesmoOut (ostream& o)
    {
        ExperimentManager::Instance().SetDesmoOut (o);
    }

// --------------------------------------------------------------------------

void Experiment::SetDesmoErr (ostream& e)
    {
        ExperimentManager::Instance().SetDesmoErr (e);
    }

// --------------------------------------------------------------------------

void Experiment::SetDesmoIn (istream& i)
    {
        ExperimentManager::Instance().SetDesmoIn (i);
    }

// --------------------------------------------------------------------------

void Experiment::ResetDesmoIO ()
    {
        ExperimentManager::Instance().ResetIO();
    }

// --------------------------------------------------------------------------

ostream& Experiment::Out()
{
    return ExperimentManager::Instance().Out();
}

// --------------------------------------------------------------------------

ostream& Experiment::Err()
{
    return ExperimentManager::Instance().Err();
}

// --------------------------------------------------------------------------
```

```
istream& Experiment::In()
{
    return ExperimentManager::Instance().In();
}

// ----------------------------------------------------------------------------
// ----------------------------------------------------------------------------

Experiment::Experiment (const String& name, const ExperimentOpts& opts)
        :   NamedObject (name),
            model       (0),
            state       (start),
            eOpts       (opts),
            eAcc        (* new ExperimentAccessory (*this))
    {
        ExperimentManager::Instance().Register (*this);
    }

// ----------------------------------------------------------------------------

Experiment::~Experiment ()
    {
        delete &eAcc;
        ExperimentManager::Instance().DeRegister (*this);
    }

// ----------------------------------------------------------------------------

String Experiment::Description () const
    {
        return "";
    }

// ----------------------------------------------------------------------------

void Experiment::Start (SimTime t)
    {
        const char* where = "Experiment::Start";

        if (!Valid())
        {
            ExperimentManager::Instance().Note (CustomGlobalErrorMessage (
                where, "invalid " + String(className),
                "Program must be aborted", "", Message::fatalError));
            return;
        }

        if (!model)
        {
            Warning (   "Attempt to start an experiment with no model",
                        where, "is ignored",
                        "create a model before starting the experiment");
            return;
        }
        if (!model->Valid())
        {
            Warning (   "Invalid main model",
                        where, "is ignored");
            return;
        }
        if (state != start)
        {
            Warning (   "Attempt to continue an experiment using 'Start'",
                        where, "'Continue' is called instead");
            Continue();
            return;
        }
        if (t <= 0.0)
            t = 0.0;    // soll sowieso nicht verdraengen

        state = running;
        ExperimentManager::Instance().StartExperiment (*this, t);
        state = stopped;
    }

// ----------------------------------------------------------------------------

void Experiment::Continue ()
    {
        const char* where = "Experiment::Continue";

        if (!Valid())
        {
```

```
            ExperimentManager::Instance().Note (CustomGlobalErrorMessage (
                where, "invalid " + String(className),
                "Program must be aborted", "", Message::fatalError));
            return;
        }

        if (!model)
        {
            Warning (   "Attempt to continue an experiment with no model",
                        where, "is ignored",
                        "create a model first an then use 'Start'");
            return;
        }
        if (!model->Valid())
        {
            Warning (   "Invalid main model",
                        where, "is ignored");
            return;
        }
        if (state != stopped)
        {
            Warning (   "Attempt to continue a not stopped experiment",
                        where, "is ignored");
            return;
        }

        state = running;
        ExperimentManager::Instance().ContinueExperiment (*this);
        state = stopped;
    }

// ---------------------------------------------------------------------------

void Experiment::Stop (SimTime dt)
    {
        const char* where = "Experiment::Stop";

        if (!Valid())
        {
            ExperimentManager::Instance().Note (CustomGlobalErrorMessage (
                where, "invalid " + String(className),
                "Program must be aborted", "", Message::fatalError));
            return;
        }

        if (!model)
        {
            Warning (   "Attempt to stop an experiment with no model",
                        where, "is ignored", "create a model first");
            return;
        }
        if (!model->Valid())
        {
            Warning (   "Invalid main model",
                        where, "is ignored");
            return;
        }
        if (state == aborted)
        {
            Warning (   "Attempt to stop an aborted experiment",
                        where, "is ignored");
            return;
        }

        if (dt < 0.0 && dt != NOW())
            dt = 0.0;

        if (dt == NOW())
            ExperimentManager::Instance().StopExperiment (*this);
        else
            (new StopSimEvent (GetModel()))->Schedule (dt);
    }

// ---------------------------------------------------------------------------

void Experiment::Report (SimTime dt)
    {
        const char* where = "Experiment::Report";

        if (!Valid())
        {
            ExperimentManager::Instance().Note (CustomGlobalErrorMessage (
                where, "invalid " + String(className),
                "Program must be aborted", "", Message::fatalError));
```

```
            return;
        }

        if (!model)
        {
            Warning (   "Attempt to create a report for an experiment with "
                        "no model",
                        where, "is ignored",
                        "create a model and use 'Start' before "
                        "calling 'Report'");
            return;
        }
        if (!model->Valid())
        {
            Warning (   "Invalid main model",
                        where, "is ignored");
            return;
        }
        if (dt <= 0.0 && dt != NOW())
            dt = 0.0;

        if (state == stopped || state == aborted || dt == NOW())
            // bei angehaltenem Experiment sofort
            ExperimentManager::Instance().Report (*this);
        else
            // sonst am Ende des Zeitraums
            (new ReportEvent (*model))->Schedule (dt);
    }

// ---------------------------------------------------------------------------

void Experiment::Reset (SimTime dt)
    {
        // SelfReport
        const char* where = "Experiment::Reset";
        if (!Valid())
        {
            ExperimentManager::Instance().Note (CustomGlobalErrorMessage (
                where, "invalid " + String(className),
                "Program must be aborted", "", Message::fatalError));
            return;
        }

        if (!model)
        {
            Warning (   "Attempt to reset an experiment with no model",
                        where, "is ignored",
                        "create a model before calling 'Reset'");
            return;
        }
        if (!model->Valid())
        {
            Warning (   "Invalid main model",
                        where, "is ignored");
            return;
        }
        if (dt <= 0.0 && dt != NOW())
            dt = 0.0;

        if (state == stopped || dt == NOW())
            // bei angehaltenem Experiment sofort
            model->Reset();
        else
            // sonst am Ende des Zeitraums
            (new ResetEvent (*model))->Schedule (dt);
    }

// ---------------------------------------------------------------------------

bool Experiment::TraceIsOn () const
    {
        const char* where = "Experiment::TraceIsOn";

        if (!Valid())
        {
            ExperimentManager::Instance().Note (CustomGlobalErrorMessage (
                where, "invalid " + String(className),
                "Program must be aborted", "", Message::fatalError));
            return false;
        }

        return eAcc.messageManager.IsOn (Message::trace);
    }
```

```
// ---------------------------------------------------------------------------

void Experiment::TraceOn (SimTime dt)
    {
        const char* where = "Experiment::TraceOn";

        if (!Valid())
        {
            ExperimentManager::Instance().Note (CustomGlobalErrorMessage (
                where, "invalid " + String(className),
                "Program must be aborted", "", Message::fatalError));
            return;
        }

        if (dt < 0.0 && dt != NOW())
        {
            Warning (   "negativ dt [" + String(dt.Time()) + ']',
                        where, "0.0 is used");
            dt = 0.0;
        }

        if (dt == NOW() || !model)
            eAcc.messageManager.SwitchOn (Message::trace);
        else
        {
            if (!model)
            {
                Warning (   "Experiment has no model",
                            where, "is ignored",
                            "create a model before you can schedule a "
                            "deferred 'TraceOn'");
                return;
            }
            if (!model->Valid())
            {
                Warning (   "Invalid main model",
                            where, "is ignored");
                return;
            }

            (new StartTraceEvent (GetModel()))->Schedule (dt);
        }
    }

// ---------------------------------------------------------------------------

void Experiment::TraceOff (SimTime dt)
    {
        const char* where = "Experiment::TraceOff";

        if (!Valid())
        {
            ExperimentManager::Instance().Note (CustomGlobalErrorMessage (
                where, "invalid " + String(className),
                "Program must be aborted", "", Message::fatalError));
            return;
        }

        if (dt < 0.0 && dt != NOW())
        {
            Warning (   "negativ dt [" + String(dt.Time()) + ']',
                        where, "0.0 is used");
            dt = 0.0;
        }

        if (dt == NOW() || !model)
            eAcc.messageManager.SwitchOff (Message::trace);
        else
        {
            if (!model)
            {
                Warning (   "Experiment has no model",
                            where, "is ignored",
                            "create a model before you can schedule a "
                            "deferred 'TraceOff'");
                return;
            }
            if (!model->Valid())
            {
                Warning (   "Invalid main model",
                            where, "is ignored");
                return;
            }
```

```
            (new EndTraceEvent (GetModel()))->Schedule (dt);
        }
    }

// ---------------------------------------------------------------------------

bool Experiment::DebugIsOn () const
    {
        const char* where = "Experiment::DebugIsOn";

        if (!Valid())
        {
            ExperimentManager::Instance().Note (CustomGlobalErrorMessage (
                where, "invalid " + String(className),
                "Program must be aborted", "", Message::fatalError));
            return false;
        }

        return eAcc.messageManager.IsOn (Message::debug);
    }

// ---------------------------------------------------------------------------

void Experiment::DebugOn (SimTime dt)
    {
        const char* where = "Experiment::DebugOn";

        if (!Valid())
        {
            ExperimentManager::Instance().Note (CustomGlobalErrorMessage (
                where, "invalid " + String(className),
                "Program must be aborted", "", Message::fatalError));
            return;
        }

        if (dt < 0.0 && dt != NOW())
        {
            Warning (   "negativ dt [" + String(dt.Time()) + ']',
                        where, "0.0 is used");
            dt = 0.0;
        }

        if (dt == NOW() || !model)
            eAcc.messageManager.SwitchOn (Message::debug);
        else
        {
            if (!model)
            {
                Warning (   "Experiment has no model",
                            where, "is ignored",
                            "create a model before you can schedule a "
                            "deferred 'DebugOn'");
                return;
            }
            if (!model->Valid())
            {
                Warning (   "Invalid main model",
                            where, "is ignored");
                return;
            }

            (new StartDebugEvent (GetModel()))->Schedule (dt);
        }
    }

// ---------------------------------------------------------------------------

void Experiment::DebugOff (SimTime dt)
    {
        const char* where = "Experiment::DebugOff";

        if (!Valid())
        {
            ExperimentManager::Instance().Note (CustomGlobalErrorMessage (
                where, "invalid " + String(className),
                "Program must be aborted", "", Message::fatalError));
            return;
        }

        if (dt < 0.0 && dt != NOW())
        {
            Warning (   "negativ dt [" + String(dt.Time()) + ']',
                        where, "0.0 is used");
            dt = 0.0;
```

```
        }

        if (dt == NOW() || !model)
            eAcc.messageManager.SwitchOff (Message::debug);
        else
        {
            if (!model)
            {
                Warning (   "Experiment has no model",
                            where, "is ignored",
                            "create a model before you can schedule a "
                            "deferred 'DebugOff'");
                return;
            }
            if (!model->Valid())
            {
                Warning (   "Invalid main model",
                            where, "is ignored");
                return;
            }

            (new EndDebugEvent (GetModel()))->Schedule (dt);
        }
    }

// --------------------------------------------------------------------------

void Experiment::SetSeed (long newSeed)
    {
        const char* where = "Experiment::SetSeed";

        if (!Valid())
        {
            ExperimentManager::Instance().Note (CustomGlobalErrorMessage (
                where, "invalid " + String(className),
                "Program must be aborted", "", Message::fatalError));
            return;
        }

        eAcc.distribManager.SeedGenerator (newSeed);
        // alle bereits erzeugten ZZ-Stroeme mit neuem Seed versorgen
        eAcc.distribManager.NewSeedAll ();
    }

// --------------------------------------------------------------------------

void Experiment::DeadlockLevel (DeadlockLevelT dl)
    {
        const char* where = "Experiment::DeadlockLevel";

        if (!Valid())
        {
            ExperimentManager::Instance().Note (CustomGlobalErrorMessage (
                where, "invalid " + String(className),
                "Program must be aborted", "", Message::fatalError));
            return;
        }

        if (dl != Off && eAcc.resourceDB.ResourceDBUsed())
        {
            Warning (   "Atempt to set DeadlockLevel after resources "
                        "has been used",
                        where, "ignored", "DeadlockLevel can set to 'Off'"
                        " or before any resources are used");
            return;
        }
        if (dl == Static)
        {
            Warning (   "DeadlockLevel 'Static' is not yet implemented",
                        where, "ignored");
            return;
        }
        eAcc.resourceDB.SetDeadlockLevel (dl);
    }

// --------------------------------------------------------------------------

Model& Experiment::GetModel ()
    {
        const char* where = "Experiment::GetModel";

        if (!Valid())
        {
            ExperimentManager::Instance().Note (CustomGlobalErrorMessage (
```

```
                where, "invalid " + String(className),
                "Program must be aborted", "", Message::fatalError));
        }

        if (!model)
            FatalError (    "Attempt to get the main model, "
                            "which does not exist yet",
                            where, "program is stopped",
                            "create a model after creating the experiment");
        if (!model->Valid())
            FatalError (    "Attempt to get the main model, "
                            "which is not valid",
                            where, "program is stopped");
        return *model;
    }

// ------------------------------------------------------------------------------

SimTime Experiment::NOW ()
    {
        return SimTime::Now();
    }

// ------------------------------------------------------------------------------

const ExperimentOpts& Experiment::GetOpts () const
    {
        return eOpts;
    }

// ------------------------------------------------------------------------------

int Experiment::NameWidth () const
    {
        return eOpts.NameWidth();
    }

// ------------------------------------------------------------------------------

int Experiment::TimeWidth () const
    {
        return eOpts.TimeWidth();
    }

// ------------------------------------------------------------------------------

int Experiment::TimePrecision () const
    {
        return eOpts.TimePrecision();
    }

// ------------------------------------------------------------------------------

int Experiment::NameNumberWidth () const
    {
        return eOpts.NameNumberWidth();
    }

// ------------------------------------------------------------------------------

SimTime Experiment::Epsilon () const
    {
        return eOpts.Epsilon();
    }

// ------------------------------------------------------------------------------

void Experiment::AddErrorOutput (Output& o)
    {
        const char* where = "Experiment::AddErrorOutput";

        if (!Valid())
        {
            ExperimentManager::Instance().Note (CustomGlobalErrorMessage (
                where, "invalid " + String(className),
                "Program must be aborted", "", Message::fatalError));
            return;
        }

        eAcc.errorManager.Add (o);
    }

// ------------------------------------------------------------------------------
```

```
void Experiment::AddDebugOutput (Output& o)
    {
        const char* where = "Experiment::AddDebugOutput";

        if (!Valid())
        {
            ExperimentManager::Instance().Note (CustomGlobalErrorMessage (
                where, "invalid " + String(className),
                "Program must be aborted", "", Message::fatalError));
            return;
        }

        eAcc.debugManager.Add (o);
    }

// ---------------------------------------------------------------------------

void Experiment::AddReportOutput (Output& o)
    {
        const char* where = "Experiment::AddReportOutput";

        if (!Valid())
        {
            ExperimentManager::Instance().Note (CustomGlobalErrorMessage (
                where, "invalid " + String(className),
                "Program must be aborted", "", Message::fatalError));
            return;
        }

        eAcc.reportManager.Add (o);
    }

// ---------------------------------------------------------------------------

void Experiment::AddTraceOutput (Output& o)
    {
        const char* where = "Experiment::AddTraceOutput";

        if (!Valid())
        {
            ExperimentManager::Instance().Note (CustomGlobalErrorMessage (
                where, "invalid " + String(className),
                "Program must be aborted", "", Message::fatalError));
            return;
        }

        eAcc.traceManager.Add (o);
    }

// ---------------------------------------------------------------------------

void Experiment::RemoveErrorOutput (Output& o)
    {
        const char* where = "Experiment::RemoveErrorOutput";

        if (!Valid())
        {
            ExperimentManager::Instance().Note (CustomGlobalErrorMessage (
                where, "invalid " + String(className),
                "Program must be aborted", "", Message::fatalError));
            return;
        }

        eAcc.errorManager.Remove (o);
    }

// ---------------------------------------------------------------------------

void Experiment::RemoveDebugOutput (Output& o)
    {
        const char* where = "Experiment::RemoveDebugOutput";

        if (!Valid())
        {
            ExperimentManager::Instance().Note (CustomGlobalErrorMessage (
                where, "invalid " + String(className),
                "Program must be aborted", "", Message::fatalError));
            return;
        }

        eAcc.debugManager.Remove (o);
    }

// ---------------------------------------------------------------------------
```

```
void Experiment::RemoveReportOutput (Output& o)
    {
        const char* where = "Experiment::RemoveReportOutput";

        if (!Valid())
        {
            ExperimentManager::Instance().Note (CustomGlobalErrorMessage (
                where, "invalid " + String(className),
                "Program must be aborted", "", Message::fatalError));
            return;
        }

        eAcc.reportManager.Remove (o);
    }

// ------------------------------------------------------------------------

void Experiment::RemoveTraceOutput (Output& o)
    {
        const char* where = "Experiment::RemoveTraceOutput";

        if (!Valid())
        {
            ExperimentManager::Instance().Note (CustomGlobalErrorMessage (
                where, "invalid " + String(className),
                "Program must be aborted", "", Message::fatalError));
            return;
        }

        eAcc.traceManager.Remove (o);
    }

// ------------------------------------------------------------------------

void Experiment::Warning (  const String& what,
                            const String& where,
                            const String& consequences,
                            const String& hint          ) const
    {
        const char* WHERE = "Experiment::Warning";

        if (!Valid())
        {
            ExperimentManager::Instance().Note (CustomGlobalErrorMessage (
                WHERE, "invalid " + String(className),
                "Program must be aborted", "", Message::fatalError));
            return;
        }

        SendMessage (CustomErrorMessage
                        (where, what, consequences, hint, Message::warning));
    }

// ------------------------------------------------------------------------

void Experiment::Error (    const String& what,
                            const String& where,
                            const String& consequences,
                            const String& hint          ) const
    {
        const char* WHERE = "Experiment::Error";

        if (!Valid())
        {
            ExperimentManager::Instance().Note (CustomGlobalErrorMessage (
                WHERE, "invalid " + String(className),
                "Program must be aborted", "", Message::fatalError));
            return;
        }

        SendMessage (CustomErrorMessage(where, what, consequences, hint));
    }

// ------------------------------------------------------------------------

void Experiment::FatalError (   const String& what,
                                const String& where,
                                const String& consequences,
                                const String& hint          ) const
    {
        const char* WHERE = "Experiment::FatalError";

        if (!Valid())
```

```
        {
            ExperimentManager::Instance().Note (CustomGlobalErrorMessage (
                WHERE, "invalid " + String(className),
                "Program must be aborted", "", Message::fatalError));
            return;
        }

        SendMessage (CustomErrorMessage
                (where, what, consequences, hint, Message::fatalError));
    }

// ---------------------------------------------------------------------------

void Experiment::SendMessage (const Message& msg) const
    {
        const char* where = "Experiment::SendMessage";

        if (!Valid())
        {
            ExperimentManager::Instance().Note (CustomGlobalErrorMessage (
                where, "invalid " + String(className),
                "Program must be aborted", "", Message::fatalError));
            return;
        }

        ExperimentManager::Instance().Note (msg, *this);
    }

// ---------------------------------------------------------------------------
```

experimm.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          experimm.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#ifndef EXPERIMENTMANAGER_H
#define EXPERIMENTMANAGER_H

// ---------------------------------------------------------------------------

class Experiment;
class Schedulable;
class Event;
class Entity;
class Process;
class SimClock;
class Scheduler;
class Model;
class Message;
class MessageManager;
class ModelComponent;
class OutputManager;
class DistribManager;
class GlobalErrorManager;
class NameCatalog;
class ResourceDB;
class DefaultExperiment;
class Reportable;
class DynamicalObject;

// ---------------------------------------------------------------------------
```

```
#include "simtime.h"
#include "interrup.h"
#include <iostream.h>

// ------------------------------------------------------------------------

class EmmCounter
{
    public:
        EmmCounter ();
        EmmCounter (const EmmCounter&);
        ~EmmCounter();
    private:
        static  unsigned long   cnt;
};

// ------------------------------------------------------------------------

static EmmCounter emmCounter;

// ------------------------------------------------------------------------

class ExperimentList;
typedef void (*memoryHandler) ();

// ------------------------------------------------------------------------

class ExperimentManager
{
        friend class EmmCounter;
    public:
        static  ExperimentManager&  Instance ();

                Experiment&     CurrentExperiment ();
                Scheduler&      CurrentScheduler ();
                Model&          CurrentModel ();
                SimClock&       CurrentSimClock ();
                MessageManager& CurrentMessageManager ();

                Scheduler&      GetScheduler        (Experiment&);
                Model&          GetModel            (Experiment&);
                SimClock&       GetSimClock         (Experiment&);
                MessageManager& GetMessageManager   (Experiment&);
                DistribManager& GetDistribManager   (Experiment&);
                ResourceDB&     GetResourceDB       (Experiment&);

                Scheduler&      GetScheduler        (const ModelComponent&);
                Model&          GetModel            (const ModelComponent&);
                SimClock&       GetSimClock         (const ModelComponent&);
                MessageManager& GetMessageManager   (const ModelComponent&);
                DistribManager& GetDistribManager   (const ModelComponent&);
                NameCatalog&    GetNameCatalog      (const ModelComponent&);
                ResourceDB&     GetResourceDB       (const ModelComponent&);

                SimTime         CurrentTime         (const ModelComponent&);
                Model&          CurrentModel        (const ModelComponent&);
                Schedulable&    Current             (const ModelComponent&);
                Event&          CurrentEvent        (const ModelComponent&);
                Entity&         CurrentEntity       (const ModelComponent&);
                Process&        CurrentProcess      (const ModelComponent&);

                GlobalErrorManager& GetGlobalErrorManager();
                Experiment&     GetDefaultExperiment();
                Model*          GetNullModel();
                Event*          GetNullEvent();
                Entity*         GetNullEntity();
                Process*        GetNullProcess();

                void            Register    (Experiment&);
                void            DeRegister  (Experiment&);
                void            Register    (Reportable&);
                void            DeRegister  (Reportable&);
                void            Register    (DynamicalObject*);
                void            DeRegister  (DynamicalObject*);

                void            ConnectToCurExp     (Model&);
                void            PrepareDeletionOf   (Model&);

                void            StartExperiment (Experiment&, SimTime t);
                void            StopExperiment (Experiment&);
                void            ContinueExperiment (Experiment&);
                void            AbortExperiment (Experiment&);
                void            Report (Experiment&);
                void            Report (Model&);
```

```
               bool            InDeletion() const;
                                   //  true, wenn EM gerade geloescht wird

               void            Note (const Message&); // an currentExperiment
               void            Note (const Message&,
                                     const Experiment& Sender);
               void            Note (const Message&,
                                     const ModelComponent& Sender);

        const   InterruptCode&  NoInterrupt() const;

               void            outOfMemory ();

               void            SetDesmoOut (ostream&);
               void            SetDesmoIn  (istream&);
               void            SetDesmoErr (ostream&);
               void            ResetIO();

               ostream&        Out();
               ostream&        Err();
               istream&        In();
    private:
        ExperimentManager ();   // Konstruktor ist privat (Singleton!)
        ExperimentManager (const ExperimentManager&);// nicht implementiert!
        ~ExperimentManager ();  // Destruktor ist privat (Singleton!)

        void                        Abort();

        static  ExperimentManager*  theSingleton; // das einzige Exemplar
        static  bool                alife;        // das einzige Exemplar lebt

                ostream*            dout;
                ostream*            derr;
                istream*            din;

        const   InterruptCode       noInterrupt;
        int                         nOfExperiments;
        GlobalErrorManager&         globalErrorManager;
        ExperimentList&             experimentList;
        DefaultExperiment*          defaultExperiment;
        Experiment*                 currentExperiment;
        bool                        inDeletionOfNullObj;
        bool                        abortFlag;
};

// ---------------------------------------------------------------------------

#endif  // EXPERIMENTMANAGER_H
```

experimm.cc

```
// ---------------------------------------------------------------------------
//
// Datei
//         experimm.cc
//
// Diplomarbeit
//
//         DESMO-C
//         Implementierung eines Simulators fuer
//         zeitdiskrete Simulation in C++
//
// Autor
//         Thomas Schniewind
//
// Datum
//         8.3.1998
//
// ---------------------------------------------------------------------------

#include "experimm.h"

#include "coroutin.h"
#include "dyobjcat.h"
#include "eacc.h"
#include "experime.h"
#include "emessage.h"
```

```
#include "messagem.h"
#include "model.h"
#include "msgtypes.h"
#include "outputm.h"
#include "ring.h"
#include "reporter.h"
#include "schedule.h"
#include "simclock.h"

#include "nullobj.h"

#include <assert.h>
#include <new.h>    // fuer memoryHandler
#include <stdlib.h> // fuer abort()

// ---------------------------------------------------------------------------
// Groesse des Speichers, der bei Speichermangel freigegeben wird

const memoryBufferSize  = 50000;

// ---------------------------------------------------------------------------

static char*            memoryBuffer;
static memoryHandler    old;

// ---------------------------------------------------------------------------

void outOfMemory ();
void outOfMemory ()
    {
        if (memoryBuffer)
        {
            // Speicher freigeben
            delete [] memoryBuffer;
            memoryBuffer = 0;
            // ExperimentManager auf Abbruch vorbereiten
            ExperimentManager::Instance().outOfMemory();
        } else
        {
            abort();
        }
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

unsigned long EmmCounter::cnt = 0;

// ---------------------------------------------------------------------------

EmmCounter::EmmCounter ()
    {
        if (cnt++ == 0)
            ExperimentManager::theSingleton = new ExperimentManager;
    }

// ---------------------------------------------------------------------------

EmmCounter::EmmCounter (const EmmCounter&)
    {
        cnt++;
    }

// ---------------------------------------------------------------------------

EmmCounter::~EmmCounter ()
    {
        if (--cnt == 0)
            delete ExperimentManager::theSingleton;
    }


// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

class ExperimentList : public Ring <Experiment>
    {};

// ---------------------------------------------------------------------------

ExperimentManager*  ExperimentManager::theSingleton = 0;
bool                ExperimentManager::alife = false;

// ---------------------------------------------------------------------------
```

```
ExperimentManager& ExperimentManager::Instance ()
    {
        assert (alife);
        return *theSingleton;
    }

// --------------------------------------------------------------------------

ExperimentManager::ExperimentManager ()
    :   dout                (&cout),
        derr                (&cerr),
        din                 (&cin),
        noInterrupt         ("NoInterrupt"),
        nOfExperiments      (0),
        globalErrorManager  (*new GlobalErrorManager (*derr)),
        experimentList      (*new ExperimentList),
        defaultExperiment   (0),
        currentExperiment   (0),
        inDeletionOfNullObj (false),
        abortFlag           (false)
    {
        alife = true;
        theSingleton = this;

        old = set_new_handler (::outOfMemory);
        if (old)
        {
            set_new_handler (old);
            memoryBuffer = 0;
        } else
            memoryBuffer = new char [50000];

        DefaultExperiment* e = new DefaultExperiment;
        defaultExperiment = e;
        ModelComponent::nullEvent       = &e->nullEvent;
        ModelComponent::nullProcess     = &e->nullProcess;
        e->eAcc.scheduler.InitCurrentObjects();
    }

// --------------------------------------------------------------------------

ExperimentManager::~ExperimentManager ()
    {
        Experiment* e;

        inDeletionOfNullObj = true;
        while ((e = experimentList.Last()) != 0)
            delete e;

        delete  &experimentList;
        delete  &globalErrorManager;
        alife = false;
        ModelComponent::nullEvent   = 0;
        ModelComponent::nullProcess = 0;

        if (memoryBuffer)
            delete [] memoryBuffer;
        set_new_handler (old);
        old = 0;
        memoryBuffer = 0;
    }

// --------------------------------------------------------------------------

void ExperimentManager::Abort ()
    {
        if (currentExperiment && currentExperiment->Valid())
        {
            *dout   << "\nCreate a report for experiment "
                    << currentExperiment->QuotedName()
                    << " before? [n] for no: ";
            char c;
            *din >> c;
            if (c != 'n' && c != 'N')
            {
                Report (*currentExperiment);
                *dout << "Report done\n";
            }
            *dout << endl;
        }
        ::abort();
        assert(0);
    }
```

```
// ----------------------------------------------------------------------------

void ExperimentManager::outOfMemory ()
    {
        // wird von ::outOfMemory aufgerufen
        abortFlag = true;
        *dout << "nearly out of memory, aborting...\a\n";
        if (currentExperiment && currentExperiment->Valid())
        {
            // Experiment anhalten
            // transferiert ggf. Kontrolle an MainCoroutine
            AbortExperiment(*currentExperiment);
            // zurueck an ::outOfMemory
            // wenn der Speicher dann ausreicht, weiter in ContiuneExperiment
        }
        else
            ::abort();
    }

// ----------------------------------------------------------------------------

Experiment& ExperimentManager::CurrentExperiment ()
    {
        assert (nOfExperiments > 0);
        assert (currentExperiment);
        return *currentExperiment;
    }

// ----------------------------------------------------------------------------

Scheduler& ExperimentManager::CurrentScheduler ()
    {
        assert (nOfExperiments > 0);
        return currentExperiment->eAcc.scheduler;
    }

// ----------------------------------------------------------------------------

Model& ExperimentManager::CurrentModel ()
    {
        assert (nOfExperiments > 0);
        return currentExperiment->eAcc.scheduler.CurrentModel();
    }


// ----------------------------------------------------------------------------

SimClock& ExperimentManager::CurrentSimClock ()
    {
        assert (nOfExperiments > 0);
        return currentExperiment->eAcc.simClock;
    }

// ----------------------------------------------------------------------------

MessageManager& ExperimentManager::CurrentMessageManager ()
    {
        assert (nOfExperiments > 0);
        return currentExperiment->eAcc.messageManager;
    }

// ----------------------------------------------------------------------------

Scheduler& ExperimentManager::GetScheduler (Experiment& e)
    {
        return e.eAcc.scheduler;
    }

// ----------------------------------------------------------------------------

Model& ExperimentManager::GetModel (Experiment& e)
    {
        return e.GetModel();
    }


// ----------------------------------------------------------------------------

SimClock& ExperimentManager::GetSimClock (Experiment& e)
    {
        return e.eAcc.simClock;
    }
```

```
// -------------------------------------------------------------------------

MessageManager& ExperimentManager::GetMessageManager (Experiment& e)
    {
        return e.eAcc.messageManager;
    }

// -------------------------------------------------------------------------

DistribManager& ExperimentManager::GetDistribManager (Experiment& e)
    {
        return e.eAcc.distribManager;
    }

// -------------------------------------------------------------------------

ResourceDB& ExperimentManager::GetResourceDB (Experiment& e)
    {
        return e.eAcc.resourceDB;
    }

// -------------------------------------------------------------------------

Scheduler& ExperimentManager::GetScheduler (const ModelComponent& m)
    {
        Experiment* e = m.GetModel().experiment;
        assert (e);
        return e->eAcc.scheduler;
    }

// -------------------------------------------------------------------------

Model& ExperimentManager::GetModel (const ModelComponent& m)
    {
        Experiment* e = m.GetModel().experiment;
        assert (e);
        return e->GetModel();
    }


// -------------------------------------------------------------------------

SimClock& ExperimentManager::GetSimClock (const ModelComponent& m)
    {
        Experiment* e = m.GetModel().experiment;
        assert (e);
        return e->eAcc.simClock;
    }

// -------------------------------------------------------------------------

MessageManager& ExperimentManager::GetMessageManager (const ModelComponent& m)
    {
        Experiment* e = m.GetModel().experiment;
        assert (e);
        return e->eAcc.messageManager;
    }

// -------------------------------------------------------------------------

DistribManager& ExperimentManager::GetDistribManager (const ModelComponent& m)
    {
        Experiment* e = m.GetModel().experiment;
        assert (e);
        return e->eAcc.distribManager;
    }

// -------------------------------------------------------------------------

NameCatalog& ExperimentManager::GetNameCatalog (const ModelComponent& m)
    {
        return *m.GetModel().nameCatalog;
    }

// -------------------------------------------------------------------------

ResourceDB& ExperimentManager::GetResourceDB (const ModelComponent& m)
    {
        Experiment* e = m.GetModel().experiment;
        assert (e);
        return e->eAcc.resourceDB;
    }

// -------------------------------------------------------------------------
```

```
SimTime ExperimentManager::CurrentTime (const ModelComponent& m)
    {
        return GetScheduler(m).CurrentTime();
    }

// ----------------------------------------------------------------------------

Model& ExperimentManager::CurrentModel (const ModelComponent& m)
    {
        return GetScheduler(m).CurrentModel();
    }

// ----------------------------------------------------------------------------

Event& ExperimentManager::CurrentEvent (const ModelComponent& m)
    {
        return GetScheduler(m).CurrentEvent();
    }

// ----------------------------------------------------------------------------

Schedulable& ExperimentManager::Current (const ModelComponent& m)
    {
        Scheduler& s = GetScheduler (m);
        if (s.CurrentEvent().IsNullEvent())
            return s.CurrentEntity();
        else
            return s.CurrentEvent();
    }

// ----------------------------------------------------------------------------

Entity& ExperimentManager::CurrentEntity (const ModelComponent& m)
    {
        return GetScheduler(m).CurrentEntity();
    }

// ----------------------------------------------------------------------------

Process& ExperimentManager::CurrentProcess (const ModelComponent& m)
    {
        return GetScheduler(m).CurrentProcess();
    }

// ----------------------------------------------------------------------------

GlobalErrorManager& ExperimentManager::GetGlobalErrorManager ()
    {
        return globalErrorManager;
    }

// ----------------------------------------------------------------------------

Experiment& ExperimentManager::GetDefaultExperiment ()
    {
        return *defaultExperiment;
    }

// ----------------------------------------------------------------------------

Model* ExperimentManager::GetNullModel ()
    {
        if (!defaultExperiment)
            return 0;
        else
            return &defaultExperiment->GetModel();
    }

// ----------------------------------------------------------------------------

Event* ExperimentManager::GetNullEvent ()
    {
        if (!defaultExperiment)
            return 0;
        else
            return &defaultExperiment->nullEvent;
    }

// ----------------------------------------------------------------------------

Entity* ExperimentManager::GetNullEntity ()
    {
        if (!defaultExperiment)
```

```
            return 0;
        else
            return &defaultExperiment->nullProcess;
    }

// ---------------------------------------------------------------------------

Process* ExperimentManager::GetNullProcess ()
    {
        if (!defaultExperiment)
            return 0;
        else
            return &defaultExperiment->nullProcess;
    }

// ---------------------------------------------------------------------------

void ExperimentManager::Register (Experiment& exp)
    {
        assert (nOfExperiments == experimentList.Size());

        currentExperiment = experimentList.Append (&exp);
        nOfExperiments++;
        exp.eAcc.messageManager.Register (globalErrorManager,
                                          Message::globalError);
    }

// ---------------------------------------------------------------------------

void ExperimentManager::DeRegister (Experiment& exp)
    {
        assert (nOfExperiments == experimentList.Size());
        assert (experimentList.Remove (&exp));
        nOfExperiments--;
        currentExperiment = experimentList.Last ();
        if (&exp == defaultExperiment)
            defaultExperiment = 0;
    }

// ---------------------------------------------------------------------------

void ExperimentManager::Register (Reportable& r)
    {
        r.GetModel().Register (r);
    }

// ---------------------------------------------------------------------------

void ExperimentManager::DeRegister (Reportable& r)
    {
        Model& m = r.GetModel();
        if (m.Valid() && &m != &r)  // Hauptmodell nicht abmelden
            m.DeRegister (r);
    }


// ---------------------------------------------------------------------------

void ExperimentManager::Register (DynamicalObject* d)
    {
        d->GetModel().dynObjCatalog.InsertDynamicalObject (d);
    }

// ---------------------------------------------------------------------------

void ExperimentManager::DeRegister (DynamicalObject* d)
    {
        d->GetModel().dynObjCatalog.removeDynamicalObject (d);
    }


// ---------------------------------------------------------------------------

void ExperimentManager::ConnectToCurExp (Model& m)
    {
        assert (currentExperiment);
        if (currentExperiment == defaultExperiment)
        {
            if (currentExperiment->model != 0)
            {   //  Fehler: Erst Experiment erzeugen, dann Modell
                Note (CustomGlobalErrorMessage (
                    "Model::Model [" + m.QuotedName() + ']',
                    "Attempt to create a model before an experiment",
                    "Program must be aborted",
```

```
                    "Create an experiment before creating a model",
                    Message::fatalError),
                    *currentExperiment);
                assert (currentExperiment->model == 0);
            }
        } else
        {
            if (currentExperiment->model != 0)
            {   //  Fehler: mehr als ein Haupt-Modell
                currentExperiment->FatalError (
                    "Attempt to create a second main model for "
                    "experiment " + currentExperiment->QuotedName(),
                    "Model::Model [" + m.QuotedName() + ']',
                    "Program must be aborted",
                    "Make the second model a submodel of the first "
                    "or create a second experiment before creating "
                    "the second model");
                assert (currentExperiment->model == 0);
            }
            currentExperiment->model = &m;
            currentExperiment->eAcc.scheduler.SetCurrentModel (m);
        }
    }

// ------------------------------------------------------------------------------

void ExperimentManager::PrepareDeletionOf (Model& m)
    {
        m.GetExperiment().eAcc.scheduler.PrepareDeletionOf (m);
    }


// ------------------------------------------------------------------------------

void ExperimentManager::StartExperiment (Experiment& e, SimTime t)
    {
        ExperimentAccessory& eAcc = e.eAcc;
        currentExperiment = &e;
        Model& m = e.GetModel();

        eAcc.scheduler.SetCurrentTime (t);
        eAcc.scheduler.SetCurrentModel (m);
        m.Reset();

        eAcc.scheduler.SetCurrentModel (m);
        m.DoInitialSchedules();
        m.DoSchedulesOfSubModels();

        eAcc.scheduler.SetCurrentModel (m);
        ContinueExperiment (e);
    }

// ------------------------------------------------------------------------------

void ExperimentManager::StopExperiment (Experiment& e)
    {
        e.state = e.stopped;
    }

// ------------------------------------------------------------------------------

void ExperimentManager::AbortExperiment (Experiment& e)
    {
        e.state = e.aborted;
    }

// ------------------------------------------------------------------------------

void ExperimentManager::ContinueExperiment (Experiment& e)
    {
        bool        gotEvent = true;
        Scheduler&  s        = e.eAcc.scheduler;

        currentExperiment = &e;
        s.InitCurrentObjects();
        while (    e.state == e.running
               && (gotEvent = s.ProcessNextEventNote()) == true)
        {
            // regelmaessig aufzurufende Tasks koennen hier bedient werden
        }
        s.InitCurrentObjects();
        if (abortFlag)
            Abort();
        if (!gotEvent)
```

```
            e.Warning ( "eventlist is empty", "Experiment::Start/Continue",
                        "experiment is stopped");
    }

// ---------------------------------------------------------------------------

void ExperimentManager::Report (Experiment& e)
    {
        String d = e.Description();
        if (d.Length() > 0)
            e.eAcc.messageManager.Note (ReportMessage (d + "\n"));

        Report (e.GetModel());
    }

// ---------------------------------------------------------------------------

void ExperimentManager::Report (Model& m)
    {
        Reporter* r = m.NewReporter();
        if (r)
        {
            m.GetExperiment().eAcc.messageManager.TakeReporter(*r);
            delete r;   // wird nicht mehr benoetigt
        }
    }

// ---------------------------------------------------------------------------

bool ExperimentManager::InDeletion() const
    {
        return inDeletionOfNullObj;
    }

// ---------------------------------------------------------------------------

const InterruptCode& ExperimentManager::NoInterrupt() const
    {
        return noInterrupt;
    }

// ---------------------------------------------------------------------------

void ExperimentManager::Note (const Message& msg)
    {
        assert (currentExperiment && currentExperiment->Valid());
        Note (msg, *currentExperiment);
    }

// ---------------------------------------------------------------------------

void ExperimentManager::Note (const Message&        msg,
                              const ModelComponent& sender)
    {
        Note (msg, sender.GetModel().GetExperiment());
    }

// ---------------------------------------------------------------------------

void ExperimentManager::Note (const Message& msg, const Experiment& sender)
    {
        Message::MessageType type = msg.Type();

        if (type == Message::error)
            if (sender.eAcc.messageManager.GetCount (type) <= 0)
                *derr << "\nAn error occured in DESMO while executing " +
                         sender.QuotedName() + "!\n"
                         "Please view your error stream(s) for further "
                         "information.\n\a\n";

        sender.eAcc.messageManager.Note (msg);

        if (type == Message::error || type == Message::globalError)
            if (msg.Code() == Message::fatalError)
            {
                *derr << "\nProgram must be aborted due to a "
                         "fatal error!\n\a\n";
                ::abort();
            }
            else if (msg.Code() == Message::normalError)
            {
                *derr << "\nExperiment must be aborted due to "
                         "an error!\n\a\n";
                AbortExperiment (*currentExperiment);
```

```
        }
    }

// ---------------------------------------------------------------------------

void ExperimentManager::SetDesmoOut (ostream& os)
{
    dout = &os;
    Coroutine::SetStreams (*dout, *derr);
}

// ---------------------------------------------------------------------------

void ExperimentManager::SetDesmoErr (ostream& os)
{
    derr = &os;
    Coroutine::SetStreams (*dout, *derr);
}

// ---------------------------------------------------------------------------

void ExperimentManager::SetDesmoIn (istream& is)
{
    din = &is;
}

// ---------------------------------------------------------------------------

void ExperimentManager::ResetIO()
{
    dout = &cout;
    derr = &cerr;
    din  = &cin;
    Coroutine::SetStreams (*dout, *derr);
}

// ---------------------------------------------------------------------------

ostream& ExperimentManager::Out()
{
    return *dout;
}

// ---------------------------------------------------------------------------

ostream& ExperimentManager::Err()
{
    return *derr;
}

// ---------------------------------------------------------------------------

istream& ExperimentManager::In()
{
    return *din;
}

// ---------------------------------------------------------------------------
```

expopts.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          expopts.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
```

```
// --------------------------------------------------------------------------

#ifndef EXPERIMENTOPTIONS_H
#define EXPERIMENTOPTIONS_H

// --------------------------------------------------------------------------

#include "simtime.h"

// --------------------------------------------------------------------------

class ExperimentOpts
{
    public:
                    ExperimentOpts (int     timewidth       = 10,
                                    int     timeprecision   =  4, // inkl. '.'
                                    int     namewidth       = 12,
                                    int     numberwidth     =  2,
                                    SimTime epsi            =  0.00001)
                        :   timeWidth       (timewidth),
                            timePrecision   (timeprecision),
                            nameWidth       (namewidth),
                            nameNumberWidth (numberwidth),
                            epsilon         (epsi)
                        {}

        int         TimeWidth()       const  { return timeWidth;         }
        int         TimePrecision()   const  { return timePrecision;     }
        int         NameWidth()       const  { return nameWidth;         }
        int         NameNumberWidth() const  { return nameNumberWidth;   }
        SimTime     Epsilon()         const  { return epsilon;           }

    protected:

    private:
        int             timeWidth;
        int             timePrecision;
        int             nameWidth;
        int             nameNumberWidth;
        SimTime         epsilon;

};

// --------------------------------------------------------------------------

#endif  // EXPERIMENTOPTIONS_H
```

histogra.h

```
// --------------------------------------------------------------------------
//
// Datei
//          histogra.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// --------------------------------------------------------------------------

#ifndef HISTOGRAM_H
#define HISTOGRAM_H

// --------------------------------------------------------------------------

#include "tally.h"  // Basisklasse

#include "str.h"

// --------------------------------------------------------------------------
```

```
typedef unsigned long   u_long;

// -------------------------------------------------------------------------

class Histogram : public Tally
{
        Histogram&  operator= (const Histogram&);   // nicht implementiert
    public:
                    Histogram ( Model&          owner,
                        const   String&         name,
                                ValueSupplier&  vs,
                                double          lower = 0.0,
                                double          upper = 0.0,
                                unsigned        cells = 1,
                                bool            showInReport = true,
                                bool            showInTrace  = false);
                    Histogram (const Histogram& objToCopy);
        virtual    ~Histogram ();

                void        ChangeParameter (   double      lower,
                                                double      upper,
                                                unsigned    cells);
        virtual void        Reset ();
        virtual void        Update();

                unsigned    Cells    () const;
                double      Lower    (unsigned cell = 1) const;
                double      Upper    () const;
                double      CellWidth() const;
                u_long      ObservationsInCell (unsigned cell) const;
                unsigned    MostFrequentedCell () const;

        virtual Reporter*   NewReporter() const;

                String      ClassName () const;

    private:
        void        checkParam (const char* where);
        void        initTable ();

        double  lower,
                upper,
                width;
        u_long  cells;
        u_long* table;
};

// -------------------------------------------------------------------------

#endif  // HISTOGRAM_H
```

histogra.cc

```
// -------------------------------------------------------------------------
//
// Datei
//          histogra.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// -------------------------------------------------------------------------

#include "histogra.h"

#include <assert.h>
#include <float.h>      // fuer DBL_MIN in Histogram::Lower()
#include "repstat.h"
```

```
// --------------------------------------------------------------------------

static const char* className = "Histogram";

// --------------------------------------------------------------------------

Histogram::Histogram (          Model&          owner,
                        const   String&         name,
                                ValueSupplier&  vs,
                                double          lo,
                                double          up,
                                unsigned        ce,
                                bool            showInReport,
                                bool            showInTrace)
    :   Tally(owner, name, vs, showInReport, showInTrace),
        lower(lo),
        upper(up),
        cells(ce),
        width(0)
    {
        const char* where = "Histogram::Histogram()";
        checkParam (where);
        assert (cells >= 1);
        table = new u_long [cells + 2];
        for (unsigned i = 0; i < cells + 2; i++)
            table [i] = 0;
    }

// --------------------------------------------------------------------------

Histogram::Histogram (const Histogram& hist)
    :   Tally(hist),
        lower(hist.lower),
        upper(hist.upper),
        cells(hist.cells),
        width(hist.width),
        table(new u_long [cells + 2])
    {
        const char* where = "Histogram::Histogram(const Histogram&)";
        checkParam (where);
        assert (cells >= 1);
        assert (table);
        for (unsigned i = 0; i < cells + 2; i++)
            table [i] = hist.table [i];
        //Reset();
    }

// --------------------------------------------------------------------------

Histogram::~Histogram ()
    {
        if (table)
            delete[] table;
    }

// --------------------------------------------------------------------------

void Histogram::checkParam (const char* where)
    {
        if (lower > upper)
        {
            Warning (   "upper is less than lower limit", where,
                        "lower and upper are exchanged");
            double temp = lower; lower = upper; upper = temp;
        }
        else if (lower == upper && cells != 1)
        {
            // Warnung: Bei gleichen Grenzen => 1 Zelle
            Warning (   "upper is equal to lower limit", where,
                        "the number of cells is set to 1");
            cells = 1;
        }
        if (cells <= 0)
        {
            // Warnung: mindestens 1 Zelle benoetigt
            Warning (   "the number of cells must be at least 1", where,
                        "the number of cells is set to 1");
            cells = 1;
        }

        assert (cells > 0);

        width = (upper - lower) / double(cells);
```

```
    }

// -------------------------------------------------------------------------

void Histogram::initTable()
    {
        assert (table);
        for (int i = 0; i < cells + 2; i++)
            table [i] = 0;
    }

// -------------------------------------------------------------------------

void Histogram::ChangeParameter (double lo, double up, unsigned ce)
    {
        const char* where = "Histogram::ChangeParameter";

        if (!valid (className, where))
            return;

        if (Observations() > 0)
        {
            // Warnung: Nur nach Reset moeglich
            Warning (   "Histogram is already used",
                        where, "-1.0 is returned", "reset it before");
            return;
        }

        assert (table);

        unsigned    oldCells = cells;
        lower   = lo;
        upper   = up;
        cells   = ce;
        checkParam (where); // brechnet width

        if (cells != oldCells)
        {
            delete[] table;
            table = new u_long [cells + 2];
            initTable();
        }
    }

// -------------------------------------------------------------------------

void Histogram::Reset()
    {
        Tally::Reset();
        if (!Valid()) return;

        assert (table);
        for (int i = 0; i < cells + 2; i++)
            table [i] = 0;
    }

// -------------------------------------------------------------------------

void Histogram::Update()
    {
        Tally::Update();

        assert (table);

        double      val = Value() - lower;
        unsigned    n;

        if (width == 0.0)
            n = (val < 0) ? 0 : (val > 0) ? 2 : 1;
        else
            n = (val < 0) ? 0 : unsigned (val / width) + 1;

        if (n > cells + 1)
            table [cells + 1]++;
        else
            table [n]++;
    }

// -------------------------------------------------------------------------

unsigned Histogram::Cells() const
    {
        return cells;
    }
```

```
// ------------------------------------------------------------------------------

double Histogram::Lower (unsigned cell) const
    {
        if (cell == 0)
            return DBL_MIN;
        else
            return lower + (cell - 1) * width;
    }

// ------------------------------------------------------------------------------

double Histogram::Upper() const
    {
        return upper;
    }

// ------------------------------------------------------------------------------

double Histogram::CellWidth() const
    {
        return width;
    }

// ------------------------------------------------------------------------------

u_long Histogram::ObservationsInCell (unsigned cell) const
    {
        const char* where = "Histogram::ObservationsInCell";

        if (!valid (className, where))
            return 0;

        if (cell > cells + 1)
        {
            // Warnung: Bereichsfehler
            Warning (   "attempt to acces an undefined cell ["
                      + String(cell) + ']',
                        where, "the value of the last cell is returned");
            cell = cells + 1;
        }

        assert (table);

        return table [cell];
    }

// ------------------------------------------------------------------------------

unsigned Histogram::MostFrequentedCell() const
    {
        u_long      max     = 0;
        unsigned    mCell   = 0;

        assert (table);
        for (int i = 0; i < cells + 2; i++)
            if (max < table [i])
            {
                max     = table [i];
                mCell   = i;
            }
        return mCell;
    }

// ------------------------------------------------------------------------------

Reporter* Histogram::NewReporter() const
    {
        return new HistogramReporter (*this);
    }

// ------------------------------------------------------------------------------

String Histogram::ClassName () const
    {
        return className;
    }

// ------------------------------------------------------------------------------
```

intdist.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          intdist.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#ifndef INTDIST_H
#define INTDIST_H

// ---------------------------------------------------------------------------

#include "distribu.h"   // Basisklasse

#include "str.h"

// ---------------------------------------------------------------------------

class IntDist : public Distribution
{
    public:
        virtual int         Sample () = 0;

        virtual    ~IntDist ();
    protected:
                    IntDist (   Model& owner,
                                const String& name = "",
                                bool showInReport = true,
                                bool showInTrace = false);

        void    swap (int &a, int &b) { int t = a; a = b; b = t; }
};

// ---------------------------------------------------------------------------

class IntDistConst : public IntDist
{
    public:
                    IntDistConst (  Model& owner,
                                    const String& name = "",
                                    int value = 0,
                                    bool showInReport = true,
                                    bool showInTrace = false);
        virtual    ~IntDistConst ();

        virtual int         Sample ();
        virtual String      GetType() const;
                                 //  liefert die Typ-Bezeichnung des ZZ-Stroms
                int         GetValue() const;
                void        ChangeParameter (int newValue);
        virtual Reporter*   NewReporter() const;
    private:
                int         value;
};

// ---------------------------------------------------------------------------

class IntDistUniform : public IntDist
{
    public:
                    IntDistUniform (    Model& owner,
                                        const String& name = "",
                                        int low     = 0,
                                        int high    = 0,
                                        bool showInReport = true,
                                        bool showInTrace = false);
        virtual    ~IntDistUniform ();
```

```
        virtual int         Sample ();
        virtual String      GetType() const;
                                //  liefert die Typ-Bezeichnung des ZZ-Stroms
                int         GetLow() const;
                int         GetHigh() const;
                void        ChangeParameter (int newLow, int newHigh);
        virtual Reporter*   NewReporter() const;
    protected:
                void        checkHiLo (const char* where);
    private:
                int         low, high;
};

// ---------------------------------------------------------------------------

class IntEmpiricalEntry;

class IntDistEmpirical : public IntDist
{
    public:
                    IntDistEmpirical (  Model& owner,
                                        const String& name = "",
                                        bool showInReport = true,
                                        bool showInTrace = false);
                    IntDistEmpirical (const IntDistEmpirical& objToCopy);
        virtual    ~IntDistEmpirical ();

        virtual int         Sample ();
        virtual String      GetType() const;
                                //  liefert die Typ-Bezeichnung des ZZ-Stroms
                void        AddEntry (int    newValue,
                                      double cumulativeFrequency);
                unsigned    CountEntries () const;
                int         GetValue (unsigned entry) const;
                                // 0 <= entry < CountEntries()
                double      GetCumulativeFrequency (unsigned entry) const;
                                // 0 <= entry < CountEntries()
        virtual Reporter*   NewReporter() const;
    private:
        unsigned            entries;
        IntEmpiricalEntry*  table;
};

// ---------------------------------------------------------------------------

class IntDistPoisson : public IntDist
{
    public:
                    IntDistPoisson (    Model& owner,
                                        const String& name = "",
                                        double mean = 0.0,
                                        bool showInReport = true,
                                        bool showInTrace = false);
        virtual    ~IntDistPoisson ();

        virtual int         Sample ();
        virtual String      GetType() const;
                            /*  liefert die Typ-Bezeichnung des ZZ-Stroms */
                double      GetMean() const;
                void        ChangeParameter (double newMean);
        virtual Reporter*   NewReporter() const;
    protected:
                void        checkMean (const char* where);
    private:
                double      mean;
};

// ---------------------------------------------------------------------------

#endif  // INTDIST_H
```

intdist.cc

```
// ---------------------------------------------------------------------------
//
// Datei
//          intdist.cc
//
```

```
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ------------------------------------------------------------------------------

#include "intdist.h"

#include <assert.h>
#include <math.h>
#include "msgdist.h"
#include "repdist.h"    // Reporter

// ------------------------------------------------------------------------------

static const char* className = "IntDist";

// ------------------------------------------------------------------------------

IntDist::IntDist (  Model& owner,
                    const String& name,
                    bool showInReport,
                    bool showInTrace)
    :   Distribution(owner, name, showInReport, showInTrace)
    {}

// ------------------------------------------------------------------------------

IntDist::~IntDist ()
    {}

// ------------------------------------------------------------------------------
// ------------------------------------------------------------------------------

IntDistConst::IntDistConst (    Model& owner,
                                const String& name,
                                int Value,
                                bool showInReport,
                                bool showInTrace)
    :   IntDist(owner, name, showInReport, showInTrace),
        value(Value)
    {
        state = Distribution::Initialized;
    }

// ------------------------------------------------------------------------------

IntDistConst::~IntDistConst ()
    {}

// ------------------------------------------------------------------------------

int IntDistConst::Sample ()
    {
        const char* where = "IntDistConst::Sample";
        if (!valid (className, where))
            return -1;
        checkSample (where);
        IncObservations();
        if (TraceIsOn())
            SendMessage (TrcDistISample (*this, value));
        return value;
    }

// ------------------------------------------------------------------------------

String IntDistConst::GetType () const
    {
        return "I-Constant";
    }

// ------------------------------------------------------------------------------

int IntDistConst::GetValue () const
    {
        return value;
```

```
    }

// ---------------------------------------------------------------------------

void IntDistConst::ChangeParameter (int newValue)
    {
        const char* where = "IntDistConst::ChangeParameter";
        if (checkParam (where))
            value = newValue;
    }

// ---------------------------------------------------------------------------

Reporter* IntDistConst::NewReporter () const
    {
        return new IntDistConstReporter (*this);
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

IntDistUniform::IntDistUniform (    Model& owner,
                                    const String& name,
                                    int Low,
                                    int High,
                                    bool showInReport,
                                    bool showInTrace)
    :   IntDist(owner, name, showInReport, showInTrace),
        low(Low),
        high(High)
    {
        const char* where = "IntDistUniform::IntDistUniform";
        state = Distribution::Initialized;
        checkHiLo (where);
    }

// ---------------------------------------------------------------------------

IntDistUniform::~IntDistUniform ()
    {}

// ---------------------------------------------------------------------------

void IntDistUniform::checkHiLo (const char* where)
    {
        if (high < low)
        {
            swap (high, low);
            SendMessage (MsgDistUnifSwap (where, *this, low, high));
        }
    }

// ---------------------------------------------------------------------------

int IntDistUniform::Sample ()
    {
        const char* where = "IntDistUniform::Sample";
        if (!valid (className, where))
            return -1;
        checkSample (where);
        IncObservations();
        int i = low + int(double(high - low + 1) * random());
        if (TraceIsOn())
            SendMessage (TrcDistISample (*this, i));
        return i;
    }

// ---------------------------------------------------------------------------

String IntDistUniform::GetType () const
    {
        return "I-Uniform";
    }

// ---------------------------------------------------------------------------

int IntDistUniform::GetLow () const
    {
        return low;
    }

// ---------------------------------------------------------------------------

int IntDistUniform::GetHigh () const
```

```
    {
        return high;
    }

// ---------------------------------------------------------------------------

void IntDistUniform::ChangeParameter (int newLow, int newHigh)
    {
        const char* where = "IntDistUniform::ChangeParameter";
        if (checkParam (where))
        {
            low     = newLow;
            high    = newHigh;
            checkHiLo (where);
        }
    }

// ---------------------------------------------------------------------------

Reporter* IntDistUniform::NewReporter () const
    {
        return new IntDistUniformReporter (*this);
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

struct IntEmpiricalEntry
{
    int                 value;
    double              probability;

                        IntEmpiricalEntry(int v = 0, double p = 0)
                            : value(v), probability(p) {}
};

// ---------------------------------------------------------------------------

IntDistEmpirical::IntDistEmpirical (    Model& owner,
                                        const String& name,
                                        bool showInReport,
                                        bool showInTrace)
    :   IntDist(owner, name, showInReport, showInTrace),
        entries(0),
        table(0)
    {}

// ---------------------------------------------------------------------------

IntDistEmpirical::IntDistEmpirical (const IntDistEmpirical& ide)
    :   IntDist (ide),
        entries (ide.entries),
        table   (new IntEmpiricalEntry [entries])
    {
        for (int i = 0; i < entries; i++)
            table [i] = ide.table [i];
    }

// ---------------------------------------------------------------------------

IntDistEmpirical::~IntDistEmpirical ()
    {
        delete[] table;
        table = 0;
        entries = 0;
    }

// ---------------------------------------------------------------------------

String IntDistEmpirical::GetType () const
    {
        return "I-Empirical";
    }

// ---------------------------------------------------------------------------

int IntDistEmpirical::Sample ()
    {
        const char* where = "IntDistEmpirical::Sample";
        if (!valid (className, where))
            return -1;
        checkSample (where);

        double   q = random();
```

```
        unsigned i = 0;

        if (!entries)
        {
            //ERROR, 0 muss ueber state abgefangen werden
            return 0;
            assert (false);
        }

        while (table[i].probability < q)
        {
            i++;
            assert(i < entries);
        }
        IncObservations();

        if (TraceIsOn())
            SendMessage (TrcDistISample (*this, table[i].value));
        return table[i].value;
    }

// ----------------------------------------------------------------------------

void IntDistEmpirical::AddEntry (int value, double probability)
    {
        const char* where = "IntDistEmpirical::AddEntry";
        if (checkParam (where))
        {
            IntEmpiricalEntry*  t = new IntEmpiricalEntry [entries + 1];

            for (unsigned n = 0; n < entries; n++)
            {
                t [n] = table [n];
                if (t [n].probability >= probability)
                {
                    SendMessage (MsgDistEmpProbOrder (where, *this,
                                                      probability,
                                                      t[n].probability));
                    delete[] t;
                    return; // Error
                }
            }
            t [entries++] = IntEmpiricalEntry (value, probability);
            if (table)
                delete[] table;
            table = t;
            if (probability >= 1.0)
                state = Distribution::Initialized;
        }
    }

// ----------------------------------------------------------------------------

unsigned IntDistEmpirical::CountEntries () const
    {
        return entries;
    }

// ----------------------------------------------------------------------------

int IntDistEmpirical::GetValue (unsigned entry) const
    {
        const char* where = "IntDistEmpirical::GetValue";
        if (entry < entries)
        {
            return table [entry].value;
        } else
        {
            SendMessage (MsgDistEmpWrongIndex (where, *this, entry, entries));
            if (entries)
                return table [entries -1].value;
            else
                return 0;
        }
    }

// ----------------------------------------------------------------------------

double IntDistEmpirical::GetCumulativeFrequency (unsigned entry) const
    {
        const char* where = "IntDistEmpirical::GetCumulativeFrequency";
        if (entry < entries)
        {
            return table [entry].probability;
```

```
        } else
        {
            // Warnung
            SendMessage (MsgDistEmpWrongIndex (where, *this, entry, entries));
            if (entries)
                return table [entries -1].probability;
            else
                return 0;
        }
    }

// ------------------------------------------------------------------------------

Reporter* IntDistEmpirical::NewReporter () const
    {
        return new IntDistEmpiricalReporter (*this);
    }

// ------------------------------------------------------------------------------
// ------------------------------------------------------------------------------

IntDistPoisson::IntDistPoisson (    Model& owner,
                                    const String& name,
                                    double Mean,
                                    bool showInReport,
                                    bool showInTrace)
    :   IntDist(owner, name, showInReport, showInTrace),
        mean(Mean)
    {
        const char* where = "IntDistPoisson::IntDistPoisson";
        state = Distribution::Initialized;
        checkMean (where);
    }

// ------------------------------------------------------------------------------

IntDistPoisson::~IntDistPoisson ()
    {}

// ------------------------------------------------------------------------------

void IntDistPoisson::checkMean (const char* where)
    {
        if (mean < 0.0)
        {
            SendMessage (MsgDistMeanNeg (where, *this, mean));
            mean = -mean;
        }
    }

// ------------------------------------------------------------------------------

int IntDistPoisson::Sample ()
    {
        const char* where = "IntDistPoisson::Sample";
        if (!valid (className, where))
            return -1;
        checkSample (where);

        double r = exp (-mean);
        double q = 1.0;
        int    m = -1;

        do
        {
            q *= random ();
            ++m;
        } while (q >= r);   // in SiFrame falsch: (q < r)!
        IncObservations();

        if (TraceIsOn())
            SendMessage (TrcDistISample (*this, m));
        return m;
    }

// ------------------------------------------------------------------------------

String IntDistPoisson::GetType () const
    {
        return "Poisson";
    }

// ------------------------------------------------------------------------------
```

```
double IntDistPoisson::GetMean () const
    {
        return mean;
    }

// ---------------------------------------------------------------------------

void IntDistPoisson::ChangeParameter (double newMean)
    {
        const char* where = "IntDistPoisson::ChangeParameter";
        if (checkParam (where))
        {
            mean = newMean;
            checkMean (where);
        }
    }

// ---------------------------------------------------------------------------

Reporter* IntDistPoisson::NewReporter () const
    {
        return new IntDistPoissonReporter (*this);
    }

// ---------------------------------------------------------------------------
```

interrup.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          interrup.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#ifndef INTERRUPTCODE_H
#define INTERRUPTCODE_H

// ---------------------------------------------------------------------------

#include "nobject.h"

#include "boolean.h"
#include "str.h"

// ---------------------------------------------------------------------------

class InterruptCode : public NamedObject
    /*  Um Prozesse zu Unterbrechen, wird an ihnen die Methode Interrupt ()
        gerufen. Als Grund fuer die Unterbrechung wird ein Objekt der Klasse
        InterruptCode uebergeben, anhand dessen der unterbrochene Prozess
        entscheiden kann, ob er unterbrochen wurde und wenn ja, wie er auf die
        Unterbrechung reagieren soll. Fuer jeden Unterbrechungsgrund muss also
        genau ein Objekt neu erzeugt werden. Zuweisung und Kopierkonstruktion
        erhalten jedoch die Identitaet, die auf der intern vergebenen Seriennr
        basiert, nicht auf den Adressen der Objekte. */
    /*  NoInterrupt () liefert das vordefinierte Objekt, das den Fall "keine
        Unterbrechung" repraesentiert.
    */
{
    public:
            InterruptCode (const String& name = "");
                // Kopierkonstruktor und Zuweisung automatisch

        static  const   InterruptCode&  NoInterrupt ();
```

```
                bool    operator== (const InterruptCode&) const;
                bool    operator!= (const InterruptCode&) const;

                unsigned long   GetCode() const;

                String  ClassName () const;

    private:
        static  unsigned long           nextCode;

                unsigned long           code;
};

// ------------------------------------------------------------------------------

#endif  // INTERRUPTCODE_H
```

interrup.cc

```
// ------------------------------------------------------------------------------
//
// Datei
//          interrup.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ------------------------------------------------------------------------------

#include "interrup.h"
#include "experimm.h"

// ------------------------------------------------------------------------------

static const char* className = "InterruptCode";

// ------------------------------------------------------------------------------

unsigned long       InterruptCode::nextCode = 0;

// ------------------------------------------------------------------------------

InterruptCode::InterruptCode (const String& name)
    :   NamedObject (name),
        code        (nextCode++)
    {}

// ------------------------------------------------------------------------------

const InterruptCode& InterruptCode::NoInterrupt ()
    {
        return ExperimentManager::Instance().NoInterrupt();
    }

// ------------------------------------------------------------------------------

bool InterruptCode::operator== (const InterruptCode& rhs) const
    {
        return code == rhs.code;
    }

// ------------------------------------------------------------------------------

bool InterruptCode::operator!= (const InterruptCode& rhs) const
    {
        return code != rhs.code;
    }

// ------------------------------------------------------------------------------
```

```
unsigned long InterruptCode::GetCode () const
    {
        return code;
    }

// ---------------------------------------------------------------------------

String InterruptCode::ClassName () const
    {
        return className;
    }

// ---------------------------------------------------------------------------
```

linevlis.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          linevlis.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#ifndef LINEAREVENTLIST_H
#define LINEAREVENTLIST_H

// ---------------------------------------------------------------------------

#include "eventlis.h"

// ---------------------------------------------------------------------------

class LinearEventNote;

// ---------------------------------------------------------------------------

class LinearEventList : public EventList
{
    public:
                LinearEventList ();
                LinearEventList (const EventList& eventList);
        virtual ~LinearEventList ();

        virtual EventList*  Clone() const;
        virtual EventList*  Clone (const EventList& source) const;
                                  // klont mit Hilfe des Copy-Konstruktors

        virtual EventNote*  FirstEventNote () const;
        virtual EventNote*  LastEventNote () const;

        virtual EventNote*  NewEventNote
                             (const SimTime& t = 0.0,
                              Event* event = 0,
                              Entity* entity = &ModelComponent::NullEntity()
                             ) const;  // Fabrikmethode
        virtual EventNote*  NewEventNote (const EventNote& note) const;
                                       // Fabrikmethode

        virtual EventNote*  NextEventNote (const EventNote* note) const;
        virtual EventNote*  PrevEventNote (const EventNote* note) const;

        virtual EventNote*  Insert        (EventNote* note);
        virtual EventNote*  InsertAsFirst (EventNote* note);
        virtual EventNote*  InsertAsLast  (EventNote* note);
        virtual EventNote*  InsertBefore  (EventNote* before,
```

```
                                         EventNote* note);
        virtual EventNote*  InsertAfter    (EventNote* after,
                                         EventNote* note);

        virtual EventNote*  Remove (EventNote* note);
        virtual EventNote*  RemoveFirst ();

        virtual void        DeleteAll ();
    private:
        LinearEventNote*    firstNote;
        LinearEventNote*    lastNote;
};

// ---------------------------------------------------------------------------

#endif  // LINEAREVENTLIST_H
```

linevlis.cc

```
// ---------------------------------------------------------------------------
//
// Datei
//          linevlis.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#include "linevlis.h"

#include "eventlis.h"
#include "boolean.h"
#include <assert.h>

// ---------------------------------------------------------------------------
// LinearEventNote
// ---------------------------------------------------------------------------

class LinearEventNote : public EventNote
{
    friend class LinearEventList;

    public:
        LinearEventNote (const  SimTime& t = 0.0,
                                Event* ev = 0,
                                Entity* en = &ModelComponent::NullEntity());
        LinearEventNote (const EventNote& eventNote);
        ~LinearEventNote ();

        virtual bool            IsScheduled () const;   // noch in der Liste?

    private:
        LinearEventNote*    next;
        LinearEventNote*    prev;
        bool                isScheduled;
};

// ---------------------------------------------------------------------------

typedef LinearEventNote* LinearEventNotePtr;    // fuer Type-Casts

// ---------------------------------------------------------------------------
// Konstruktoren / Destruktor

LinearEventNote::LinearEventNote (const SimTime& t,
                                        Event* ev,
                                        Entity* en) // Model!!
    :   EventNote   (t, ev, en),
```

```
        next        (0),
        prev        (0),
        isScheduled (false)
    {}

// ---------------------------------------------------------------------------

LinearEventNote::LinearEventNote (const EventNote& eventNote)
    :   EventNote   (eventNote),
        next        (0),
        prev        (0),
        isScheduled (false)
    {}

// ---------------------------------------------------------------------------

LinearEventNote::~LinearEventNote ()
    {}

// ---------------------------------------------------------------------------
// Methoden

bool LinearEventNote::IsScheduled () const
    {
        return isScheduled;
    }

// ---------------------------------------------------------------------------
// LinearEventList
// ---------------------------------------------------------------------------

// ---------------------------------------------------------------------------
// Konstruktoren / Destruktor

LinearEventList::LinearEventList ()
    :   firstNote(0), lastNote(0)
    {}

// ---------------------------------------------------------------------------

LinearEventList::LinearEventList (const EventList& source)
    :   firstNote(0), lastNote(0)
    {
        EventNote*  note    = source.FirstEventNote ();

        while (note)
        {
            InsertAsLast (NewEventNote (*note));
            note = NextEventNote (note);
        }
    }

// ---------------------------------------------------------------------------

LinearEventList::~LinearEventList ()
    {
        DeleteAll();
    }

// ---------------------------------------------------------------------------
// Methoden

EventList* LinearEventList::Clone() const
    {
        return new LinearEventList;
    }

// ---------------------------------------------------------------------------

EventList* LinearEventList::Clone (const EventList& source) const
    {
        return new LinearEventList (source);
    }


// ---------------------------------------------------------------------------

EventNote* LinearEventList::FirstEventNote () const
    {
        return firstNote;
    }

// ---------------------------------------------------------------------------
```

```
EventNote* LinearEventList::LastEventNote () const
    {
        return lastNote;
    }

// ----------------------------------------------------------------------------

EventNote* LinearEventList::NewEventNote (const SimTime& t,
                                                Event* event,
                                                Entity* entity) const
    // Fabrikmethode
    {
        return new LinearEventNote (t, event, entity);
    }

// ----------------------------------------------------------------------------

EventNote* LinearEventList::NewEventNote (  const EventNote& note) const
    // Fabrikmethode
    {
        return new LinearEventNote (note);
    }

// ----------------------------------------------------------------------------

EventNote* LinearEventList::NextEventNote (const EventNote* note) const
    {
        if (!note)  return 0;
        else        return LinearEventNotePtr(note)->next;
    }

// ----------------------------------------------------------------------------

EventNote* LinearEventList::PrevEventNote (const EventNote* note) const
    {
        if (!note)  return 0;
        else        return LinearEventNotePtr(note)->prev;
    }

// ----------------------------------------------------------------------------

EventNote* LinearEventList::Insert (EventNote* note)
    {
        if (!note) return 0;
        if (!firstNote) // firstNote == lastNote == 0 => Liste ist leer
            return InsertAsFirst (note);

        assert (lastNote);  // wenn es ein erstes gibt, dann auch ein letztes

        LinearEventNote* temp = firstNote;
        while (temp && (*temp <= *note))
            temp = temp->next;

        // hier gilt: temp == 0 oder das erste, das groesser als note ist

        if (temp)   // (temp != 0) => (note vor temp eingefuegen)
            return InsertBefore (temp, note);
        else        // (temp == 0) => (*note > *lastNote)
                    //             => note nach lastNote einfuegen
            return InsertAfter (lastNote, note);
    }

// ----------------------------------------------------------------------------

EventNote* LinearEventList::InsertAsFirst (EventNote* note)
    {
        if (!note) return 0;
        if (firstNote)
            // Liste ist nicht leer
            return InsertBefore (firstNote, note);
        else
        {
            firstNote = lastNote = LinearEventNotePtr(note);
            firstNote->isScheduled = true;
            return firstNote;
        }
    }

// ----------------------------------------------------------------------------

EventNote* LinearEventList::InsertAsLast (EventNote* note)
    {
        if (!note) return 0;
        if (lastNote)
```

```
            // Liste ist nicht leer
            return InsertAfter (lastNote, note);
        else
        {
            firstNote = lastNote = LinearEventNotePtr(note);
            firstNote->isScheduled = true;
            return firstNote;
        }
    }

// ---------------------------------------------------------------------------

EventNote* LinearEventList::InsertBefore (EventNote* Before, EventNote* note)
    {
        if (!note || !Before) return 0;
        if (*note > *Before) return 0;
        LinearEventNote* lnote  = LinearEventNotePtr(note);
        LinearEventNote* before = LinearEventNotePtr(Before);
        if (!before->isScheduled) return 0;
        assert (firstNote);

        // OK: lnote vor before einfuegen
        if (before->prev)   // before hat einen Vorgaenger
            before->prev->next = lnote;
        else
        {                   // before muss das erste gewesen sein
            assert (before == firstNote);
            firstNote = lnote;
        }
        lnote->prev     = before->prev;
        lnote->next     = before;
        before->prev    = lnote;
        lnote->isScheduled = true;
        return lnote;
    }

// ---------------------------------------------------------------------------

EventNote* LinearEventList::InsertAfter (EventNote* After, EventNote* note)
    {
        if (!note || !After) return 0;
        if (*note < *After) return 0;

        LinearEventNote* lnote  = LinearEventNotePtr(note);
        LinearEventNote* after  = LinearEventNotePtr(After);
        if (!after->isScheduled) return 0;
        assert (lastNote);

        // OK: lnote hinter after einfuegen
        if (after->next)    // after hat einen Nachfolger
            after->next->prev = lnote;
        else
        {                   // after muss das letzte gewesen sein
            assert (after == lastNote);
            lastNote = lnote;
        }
        lnote->next = after->next;
        lnote->prev = after;
        after->next = lnote;
        lnote->isScheduled = true;
        return lnote;
    }

// ---------------------------------------------------------------------------

EventNote* LinearEventList::Remove (EventNote* note)
    {
        if (!note) return 0;
        if (!note->IsScheduled()) return 0;
        LinearEventNote* lnote = LinearEventNotePtr(note);

        if (lnote->prev)
            lnote->prev->next = lnote->next;
        else
        {
            assert (lnote == firstNote);
            firstNote = lnote->next;
        }

        if (lnote->next)
            lnote->next->prev = lnote->prev;
        else
        {
            assert (lnote == lastNote);
```

```
            lastNote = lnote->prev;
        }

        lnote->prev =
        lnote->next = 0;
        lnote->isScheduled = false;
        return lnote;
    }

// ----------------------------------------------------------------------------

EventNote* LinearEventList::RemoveFirst()
    {
        if (!firstNote)
            return 0;
        else
            return Remove (firstNote);
    }

// ----------------------------------------------------------------------------

void LinearEventList::DeleteAll()
    {
        EventNote*  temp;
        while ((temp = RemoveFirst()) != 0)
            {   delete temp; };
    }

// ----------------------------------------------------------------------------
```

message.h

```
// ----------------------------------------------------------------------------
//
// Datei
//          message.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ----------------------------------------------------------------------------
//
// Beschreibung
//
//          alle Nachrichtentypen, die vom System unterschieden werden
//
// ----------------------------------------------------------------------------

#ifndef MESSAGE_H
#define MESSAGE_H

// ----------------------------------------------------------------------------

#include "simtime.h"
#include "str.h"

// ----------------------------------------------------------------------------

class Schedulable;
class Event;
class Entity;
class Process;
class Model;
class ModelComponent;
class Experiment;

// ----------------------------------------------------------------------------

class Message
```

```
    /*  Abstrakte Oberklasse fuer alle Meldugnen im System. Die rein virtuelle
        Methode 'Description' liefert den Meldungstext. */
    /*  Ueber MessageType kann die Art der Meldung bestimmt werden, so dass
        bestimmte Meldungen auch nur in bestimmten Reports erscheinen. */
    /*  Globale Fehler treten auf, wenn Versucht wird aus einem Experiment
        Objekte eines anderen Experiments z.B. vorzumerken, oder noch kein
        Experiment existiert, somit auch kein Ausgabekanal fuer die Meldung.
    */
{
    public:
        enum MessageType {// alle von 0 bis custom nummerieren!
                            globalError = 0,
                            error       = 1,
                            report      = 2,
                            trace       = 3,
                            debug       = 4,
                            custom      = 5};
                          // custom muss immer der letzte sein

        enum CodeType {     normal,
                            switchOn, switchOff,
                            fatalError, normalError, warning,
                            descriptionAsBox
                        };

                Message (MessageType, CodeType ct = normal);
        virtual ~Message ();

                SimTime     Time () const;  // Zeitpunkt der Nachricht
        const   Event&      GetEvent() const;
        const   Entity&     GetEntity() const;
        const   Process&    GetProcess() const;
        const   Model&      GetModel() const;
        const   Experiment& GetExperiment() const;

                MessageType Type() const;
                int         Code () const;
                String      CodeText () const;

        virtual String      Description () const;
        virtual String      Location () const;
        virtual String      Consequences () const;
        virtual String      Hint () const;
    protected:
                String      Quote (const String&) const;
                String      NameAndModel (const ModelComponent&) const;
                String      TxtDtToAt (const SimTime& dt) const;
                String      TxtTimeToAt (const SimTime& t) const;
                String      TxtItselfIfCurrent (const Schedulable& s) const;
    private:
        MessageType             messageType;    // Art der Nachricht
        int                     code;
};

// ---------------------------------------------------------------------------

#endif  // MESSAGE_H
```

message.cc

```
// ---------------------------------------------------------------------------
//
// Datei
//          message.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------
```

```
#include "message.h"

#include "event.h"
#include "experimm.h"
#include "model.h"
#include "modelcom.h"
#include "schedule.h"
#include "simclock.h"
#include "str.h"
#include "strstr.h"
#include "text.h"

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------
// Klasse Message:

Message::Message (MessageType mt, CodeType ct)
    :   messageType (mt),
        code(ct)
    {}

// ---------------------------------------------------------------------------

Message::~Message () {};    // virtueller Destruktor der Basisklasse

// ---------------------------------------------------------------------------

SimTime Message::Time () const
    {   return ExperimentManager::Instance().CurrentSimClock().Time(); }

// ---------------------------------------------------------------------------

const Event& Message::GetEvent () const
    { return ExperimentManager::Instance().CurrentScheduler().CurrentEvent();}

// ---------------------------------------------------------------------------

const Entity& Message::GetEntity () const
    {return ExperimentManager::Instance().CurrentScheduler().CurrentEntity();}

// ---------------------------------------------------------------------------

const Process& Message::GetProcess () const
   {return ExperimentManager::Instance().CurrentScheduler().CurrentProcess();}

// ---------------------------------------------------------------------------

const Model& Message::GetModel () const
    {   return ExperimentManager::Instance().CurrentModel(); }

// ---------------------------------------------------------------------------

const Experiment& Message::GetExperiment () const
    {   return ExperimentManager::Instance().CurrentExperiment(); }

// ---------------------------------------------------------------------------

Message::MessageType Message::Type () const
    {   return messageType; }

// ---------------------------------------------------------------------------

int Message::Code () const
    {   return code; }

// ---------------------------------------------------------------------------

String Message::Quote (const String& s) const
    {
        strstream ss;
        ss << '\'' << s << '\'' << ends;
        return ss;
    }

// ---------------------------------------------------------------------------

String Message::NameAndModel (const ModelComponent& mc) const
    {
        strstream ss;
        ss << '\'' << mc.Name() << "' of model '" << mc.GetModel().Name()
           << '\'' << ends;
        return ss;
    }
```

```
// --------------------------------------------------------------------------

String Message::Description() const
    {   return ""; }

// --------------------------------------------------------------------------

String Message::Location() const
    {   return ""; }

// --------------------------------------------------------------------------

String Message::Consequences() const
    {   return ""; }

// --------------------------------------------------------------------------

String Message::Hint() const
    {   return ""; }

// --------------------------------------------------------------------------

String Message::CodeText () const
    {
        switch (code)
        {
            case fatalError:
                return "Fatal Error";
            case normalError:
                return "Error";
            case warning:
                return "Warning";
            default:
                return "";
        };
    }

// --------------------------------------------------------------------------

String Message::TxtDtToAt (const SimTime& dt) const
    {
        strstream ss;
        if (dt == dt.Now())
            ss << txtNOW;
        else if (dt == 0.0)
            ss << txtNow;
        else
            ss << "at " << (dt + Time());
        ss << ends;
        return ss;
    }

// --------------------------------------------------------------------------

String Message::TxtTimeToAt (const SimTime& t) const
    {
        strstream ss;
        if (t == Time())
            ss << txtNow;
        else
            ss << "at " << (t);
        ss << ends;
        return ss;
    }

// --------------------------------------------------------------------------

String Message::TxtItselfIfCurrent (const Schedulable& s) const
    {
        if (s.IsCurrent())
            if (GetEvent().IsNullEvent())
                // Prozess-orientiert
                return txtItself;
            else
                return txtIt;
        else
            return s.QuotedName();
    }

// --------------------------------------------------------------------------
```

messaged.h

```
// --------------------------------------------------------------------------
//
// Datei
//          messaged.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// --------------------------------------------------------------------------
//
// Beschreibung
//
//          enthaelt fuer jeden in "message.h" deklarierten
//          Nachrichten-Typ ein Methodenpaar Note und Distribute
//          sowie eine Empfaengerliste Distribute verteilt eine
//          empfangene Nachricht an alle mit Register eingetragenen
//          Empfaenger Die Default-Implementation von Note ruft
//          Distribute auf mit leerer Default-Implementation
//
// --------------------------------------------------------------------------

#ifndef MESSAGEDISTRIBUTOR_H
#define MESSAGEDISTRIBUTOR_H

// --------------------------------------------------------------------------

#include "messager.h"
#include "message.h"    // fuer MessageType

// --------------------------------------------------------------------------

class ReceiverList;
class Reporter;

// --------------------------------------------------------------------------

class MessageDistributor : public MessageReceiver
{
    public:
                        MessageDistributor ();
        virtual         ~MessageDistributor ();

                        // fuer Nachrichtentyp an- und abmelden:
                void    Register    (MessageReceiver& r,
                                     Message::MessageType t);
                void    DeRegister  (MessageReceiver& r,
                                     Message::MessageType t);
                        // fuer alle abmelden
                void    DeRegister  (MessageReceiver& r);

                int     NoOfReceiver (Message::MessageType t) const;

        virtual void    Note (const Message& msg);
        virtual void    TakeReporter (Reporter& r);

    protected:
                void    Distribute (const Message& msg);
                void    Distribute (Reporter& r);

    private:
        ReceiverList*   receiverList;   // interessierte Empfaenger
        int             noOfReceiver;   // Anzahl der Empfaenger
};

// --------------------------------------------------------------------------

#endif  // MESSAGEDISTRIBUTOR_H
```

messaged.cc

```
// ---------------------------------------------------------------------------
//
// Datei
//          messaged.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#include "messaged.h"
#include "messager.h"
#include "message.h"
#include "ring.h"

// ---------------------------------------------------------------------------

class ReceiverList : public Ring <MessageReceiver>
{};

// ---------------------------------------------------------------------------

MessageDistributor::MessageDistributor ()
    :   receiverList(new ReceiverList [Message::custom+1]),
        noOfReceiver (0)
    {}

// ---------------------------------------------------------------------------

MessageDistributor::~MessageDistributor ()
    {
        delete[] receiverList;
    }

// ---------------------------------------------------------------------------

void MessageDistributor::Register (MessageReceiver& r, Message::MessageType t)
    {
        receiverList[t].Append (&r);
        ++noOfReceiver;
    }

// ---------------------------------------------------------------------------

void MessageDistributor::DeRegister (MessageReceiver&     r,
                                     Message::MessageType t)
    {
        if (receiverList [t].Find (&r))
        {
            receiverList [t].Dequeue();
            --noOfReceiver;
        }
    }

// ---------------------------------------------------------------------------

void MessageDistributor::DeRegister (MessageReceiver& r)
    {
        int i;
        for (i = 0; i <= Message::custom; ++i)
            DeRegister (r, Message::MessageType (i));
    }

// ---------------------------------------------------------------------------

int MessageDistributor::NoOfReceiver (Message::MessageType t) const
    {   return receiverList [t].Size(); }

// ---------------------------------------------------------------------------
// Messages
```

```
void MessageDistributor::Note (const Message& msg)
    {   Distribute (msg); }

// ---------------------------------------------------------------------------

void MessageDistributor::TakeReporter (Reporter& r)
    {   Distribute (r); }

// ---------------------------------------------------------------------------

void MessageDistributor::Distribute (const Message& msg)
    {
        Message::MessageType    type = msg.Type();
        MessageReceiver*        r = receiverList [type].First();

        for (int i = receiverList [type].Size(); i > 0; --i)
        {
            r->Note (msg);
            r = receiverList [type].Next();
        }
    }

// ---------------------------------------------------------------------------

void MessageDistributor::Distribute (Reporter& reporter)
    {
        Message::MessageType    type = Message::report;
        MessageReceiver*        r = receiverList [type].First();

        for (int i = receiverList [type].Size(); i > 0; --i)
        {
            r->TakeReporter (reporter);
            r = receiverList [type].Next();
        }
    }

// ---------------------------------------------------------------------------
```

messagem.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          messagem.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#ifndef MESSAGEMANAGER_H
#define MESSAGEMANAGER_H

// ---------------------------------------------------------------------------

#include "messaged.h"

#include "boolean.h"

// ---------------------------------------------------------------------------

class MessageManager : public MessageDistributor
{
    public:
        MessageManager ();

        virtual void    Note (const Message& msg);

                bool    IsOn (Message::MessageType t) const;
```

```
                unsigned
                long    GetCount (Message::MessageType t) const;

                void    Skip (  unsigned             skip = 1,
                                Message::MessageType t    = Message::trace);

                void    SwitchOn (Message::MessageType t);
                void    SwitchOff(Message::MessageType t);

    private:
        bool            isOn [Message::custom+1];
        int             skip [Message::custom+1];
        unsigned long   count[Message::custom+1];
};

// ------------------------------------------------------------------------------

#endif  // MESSAGEMANAGER_H
```

messagem.cc

```
// ------------------------------------------------------------------------------
//
// Datei
//          messagem.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ------------------------------------------------------------------------------

#include "messagem.h"

// ------------------------------------------------------------------------------

MessageManager::MessageManager()
    {
        for (int type = 0; type <= Message::custom; ++type)
        {
            isOn [type] = false;
            skip [type] = 0;
            count[type] = 0;
        }
        isOn [Message::globalError] = true;
        isOn [Message::error] = true;
        isOn [Message::report]= true;
    }


// ------------------------------------------------------------------------------

void MessageManager::Note (const Message& msg)
    {
        Message::MessageType type = msg.Type();

        if (IsOn (type))
            if (skip [type]-- <= 0)
            {
                skip [type] = 0;
                count[type]++;
                Distribute (msg);
            }
    }

// ------------------------------------------------------------------------------

bool MessageManager::IsOn (Message::MessageType t) const
    {
```

```
        return isOn [t];
    }

// -----------------------------------------------------------------------------

unsigned long MessageManager::GetCount (Message::MessageType type) const
    {
        return count [type];
    }

// -----------------------------------------------------------------------------

void MessageManager::Skip (unsigned s, Message::MessageType type)
    {
        skip [type] += s;
    }

// -----------------------------------------------------------------------------

void MessageManager::SwitchOn (Message::MessageType type)
    {
        if (!IsOn (type))
        {
            skip [type] = 0;
            isOn [type] = true;
            Note (Message (type, Message::switchOn));
        }
    }

// -----------------------------------------------------------------------------

void MessageManager::SwitchOff (Message::MessageType type)
    {
        if (IsOn (type))
        {
            Note (Message (type, Message::switchOff));
            isOn [type] = false;
        }
    }

// -----------------------------------------------------------------------------
```

messager.h

```
// -----------------------------------------------------------------------------
//
// Datei
//          messager.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// -----------------------------------------------------------------------------

#ifndef MESSAGERECEIVER_H
#define MESSAGERECEIVER_H

// -----------------------------------------------------------------------------

class Message;
class Reporter;

// -----------------------------------------------------------------------------

class MessageReceiver
{
    public:
        virtual void    Note (const Message& msg) {};
        virtual void    TakeReporter (Reporter& r) {};
```

```
};

// -------------------------------------------------------------------------

#endif  // MESSAGERECEIVER_H
```

model.h

```
// -------------------------------------------------------------------------
//
// Datei
//          model.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// -------------------------------------------------------------------------

#ifndef MODEL_H
#define MODEL_H

// -------------------------------------------------------------------------

#include "reportab.h"
#include "boolean.h"
#include "str.h"

// -------------------------------------------------------------------------

class Experiment;
class ModelList;
class ReportableList;
class Reporter;
class Scheduler;
class NameCatalog;
class DynObjCatalog;

// -------------------------------------------------------------------------

class Model : public Reportable //  ModelComponent
{
        friend class ExperimentManager;

                Model (const Model&);       // nicht implementiert
        Model&  operator= (const Model&);   // nicht implementiert

    public:
                Model (Model* owner, const String& name,
                       bool showInReport = true,
                       bool showInTrace  = true);
                    /*  owner ist das Modell, das dieses als Submodell
                        benutzt. Ein Selbstverweis oder 0 zeigen an, dass
                        es sich um das Oberste Modell handelt. */

        virtual ~Model ();

        virtual void            Reset();

                bool            Connected() const;
                    /*  true => Model ist mit Experiment verbunden */
        virtual Experiment&     GetExperiment() const;

                QueueOption     GetQueueOption () const;
        virtual void            SetQueueOption (QueueOption);
                    /*  Setzt auch QueueOption der Submodelle,
                        kann jedoch von Unterklassen redefiniert werden,
                        wenn in bestimmten (Unter-)Modellen die
                        MultipleQueue-Option benoetigt wird. */
```

```
                void            Report ();
                    /*  Gibt das Modell unmittelbar in den Report-
                        Kanal aus. */

        virtual String          Description () const;

        virtual bool            CheckCompatibility (
                                    const ModelComponent* m1,
                                    const ModelComponent* m2) const;
                    /*  liefert true, wenn m1 und m2 kompatibel sind.
                        Standard: this == &m2->GetModel()
                        (denn m1 hat zum Aufruf dieser Methode gefuehrt)
                        Kann ueberschrieben werden, um eine eigene Pruefung
                        zu implementieren, welche Komponenten innerhalb
                        des Modells verarbeitet werden koennen
                        (z.B mit RTTI).
                        Hauptaugenmerk sollte auf m2 liegen. */

                bool            InDestruction() const;
                    /*  true, wenn das Modell gerade destruiert wird */

        virtual Reporter*       NewReporter() const;
                String          ClassName () const;
    protected:
        virtual void    DoInitialSchedules ();
                    /*  Hier werden die ersten Simulationsobjekte in die
                        Ereignisliste eingetragen. Die Methode wird vom
                        System automatisch aufgerufen, nachdem das Experiment
                        ueber 'Start' angestossen wurde. */

    private:
                void            Connect     (Experiment& e);
                                //  macht e zum Experiment des Modells
                void            Connect     (Model& subModel);
                                /*  wird vom Submodell gerufen, um sich in
                                    die Liste der Submodelle einzureihen */
                void            Disconnect  (Model& subModel);
                                //  Entfernen des Submodells

                void            Register    (Reportable&); // fuer die
                void            DeRegister  (Reportable&); // Reportliste

                void            DoSchedulesOfSubModels ();

                Experiment*     experiment;
                bool            inDestruction;
                ModelList&      subModels;
                ReportableList& reportables;
                QueueOption     queueOption;
                NameCatalog*    nameCatalog;
                DynObjCatalog&  dynObjCatalog;
};

// ------------------------------------------------------------------------------

#endif
```

model.cc

```
// ------------------------------------------------------------------------------
//
// Datei
//         model.cc
//
// Diplomarbeit
//
//         DESMO-C
//         Implementierung eines Simulators fuer
//         zeitdiskrete Simulation in C++
//
// Autor
//         Thomas Schniewind
//
// Datum
//         8.3.1998
//
// ------------------------------------------------------------------------------
```

```
#include "model.h"

#include "dyobjcat.h"
#include "experime.h"
#include "experimm.h"
#include "namecat.h"
#include "repmodel.h"
#include "reportab.h"
#include "reportal.h"   // ReportableList
#include "ring.h"
#include "schedule.h"
#include <assert.h>

// ---------------------------------------------------------------------------

class ModelList : public Ring<Model> {};

// ---------------------------------------------------------------------------

static const char* className = "Model";

// ---------------------------------------------------------------------------

Model::Model (Model* owner, const String& name,
              bool showInReport, bool showInTrace)
    :   Reportable      ((owner ? *owner : *this), name,
                         showInReport, showInTrace),
        experiment      (0),
        inDestruction   (false),
        subModels       (*new ModelList),
        reportables     (*new ReportableList),
        queueOption     (OnlyOneQueue),
        nameCatalog     (0),        // s.u.
        dynObjCatalog   (*new DynObjCatalog)
    {
        if (&GetModel() == this)
        {
            // als Hauptmodell mit laufendem Experiment verbinden
            ExperimentManager::Instance().ConnectToCurExp (*this);
            experiment = &ExperimentManager::Instance().CurrentExperiment();
        }
        else
        {
            // als Submodell mit owner verbinden
            Model& myOwner = GetModel();
            myOwner.Connect (*this);
            experiment = myOwner.experiment;
        }

        nameCatalog = new NameCatalog ( experiment->NameWidth(),
                                        experiment->NameNumberWidth());
        Reset();
    }

// ---------------------------------------------------------------------------

Model::~Model ()
    {
        Model& owner = GetModel();
        if (&owner != this)
        {
            owner.Disconnect (*this);
            owner.DeRegister (*this);
        }
        ExperimentManager::Instance().PrepareDeletionOf (*this);
        inDestruction = true;
        delete nameCatalog;
        delete &dynObjCatalog;
        delete &subModels;
        delete &reportables;
    }

// ---------------------------------------------------------------------------

void Model::Reset()
    {
        Reportable::Reset();
        if (!Valid()) return;

        Reportable* r = reportables.First();

        for (int i = reportables.Size(); i > 0; i--)
        {
            assert (r);
```

```
            r->Reset();
            r = reportables.Next();
        }
    }

// ---------------------------------------------------------------------------

bool Model::Connected() const
    {
        return (experiment ? true : false);
    }

// ---------------------------------------------------------------------------

void Model::DoInitialSchedules ()
    {}

// ---------------------------------------------------------------------------

Experiment& Model::GetExperiment() const
    {
        // darf erst gerufen werden, wenn bereits mit Experiment verbunden
        assert (experiment);
        return *experiment;
    }

// ---------------------------------------------------------------------------

void Model::Connect (Experiment& e)
    {
        // Darf nur einmal einem Experiment zugeordnet werden!
        assert (experiment == 0);

        experiment = &e;

        // Submodelle verbinden
        int i;
        subModels.First();
        for (i = 1; i <= subModels.Size(); i++)
        {
            subModels.Current()->Connect(e);
            subModels.Next();
        }
    }

// ---------------------------------------------------------------------------

void Model::Connect (Model& subModel)
    {
        assert (!subModels.Find (&subModel));
        subModels.Append(&subModel);
        subModel.experiment = experiment;
    }

// ---------------------------------------------------------------------------

void Model::Disconnect (Model& subModel)
    {
        subModels.Remove(&subModel);
    }

// ---------------------------------------------------------------------------

void Model::DoSchedulesOfSubModels ()
    {
        Scheduler& scheduler = ExperimentManager::Instance().
                                GetScheduler (*experiment);
        int i;
        Model* m = subModels.First();
        for (i = 1; i <= subModels.Size(); i++)
        {
            scheduler.SetCurrentModel (*m);
            m->DoInitialSchedules();
            m->DoSchedulesOfSubModels();
            m = subModels.Next();
        }
    }

// ---------------------------------------------------------------------------

void Model::Register (Reportable& r)
    {
        if (&r != this)
            //  Destruktoren werden wahrscheinlich in umgekehrter
```

```
            //  Reihenfolge aufgerufen => wie Stack behandeln
            reportables.Push (&r);
    }

// ----------------------------------------------------------------------------

void Model::DeRegister (Reportable& r)
    {
        if (&r != this)
            if (!reportables.Remove (&r))
                assert (false); // Warnung: Reportable war nicht registriert
    }

// ----------------------------------------------------------------------------

QueueOption Model::GetQueueOption () const
    {
        return queueOption;
    }

// ----------------------------------------------------------------------------

void Model::SetQueueOption (QueueOption qo)
    {
        const char* where = "Model::SetQueueOption";

        if (!valid (className, where))
            return;

        queueOption = qo;

        Model* m = subModels.First();

        for (int i = subModels.Size(); i > 0; --i)
        {
            m->SetQueueOption (qo);
            m = subModels.Next();
        }
    }

// ----------------------------------------------------------------------------

void Model::Report ()
    {
        const char* where = "Model::Report";

        if (!valid (className, where))
            return;

        ExperimentManager::Instance().Report (*this);
    }

// ----------------------------------------------------------------------------

String Model::Description () const
    {
        return "";
    }

// ----------------------------------------------------------------------------

bool Model::CheckCompatibility (const ModelComponent* m1,
                                const ModelComponent* m2) const
    {
        return this == &m2->GetModel();
    }

// ----------------------------------------------------------------------------

bool Model::InDestruction () const
    {
        return inDestruction;
    }


// ----------------------------------------------------------------------------

Reporter* Model::NewReporter () const
    {
        return new ModelReporter (*this, reportables);
    }


// ----------------------------------------------------------------------------
```

```
String Model::ClassName () const
    {
        return className;
    }

// -----------------------------------------------------------------------------
```

modelcom.h

```
// -----------------------------------------------------------------------------
//
// Datei
//          modelcom.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// -----------------------------------------------------------------------------
#ifndef MODELCOMPONENT_H
#define MODELCOMPONENT_H

// -----------------------------------------------------------------------------

#include "nobject.h"    // Basisklasse

#include "message.h"
#include "simtime.h"
#include "str.h"

// -----------------------------------------------------------------------------

class Schedulable;
class Entity;
class Event;
class Experiment;
class ExperimentOpts;
class Message;
class Model;
class Process;
class Scheduler;

// -----------------------------------------------------------------------------

enum QueueOption {      // Option fuer Warteschlangen
        OnlyOneQueue,   // ein Entity kann nur in einer WS zur Zeit warten
        MultipleQueue   // ein Entity kann in mehreren WS gleichzeitig warten,
                        // jedoch nicht zweimal in der selben
    };

// -----------------------------------------------------------------------------

class ModelComponent : public NamedObject
    /*  ModelComponent ist Oberklasse fuer alle Objekte, die zu einem
        Modell gehoeren. Durch die Zuordnung laesst sich pruefen, ob
        Komponenten kompatibel sind.
    */
{
        friend class ExperimentManager;

    public:
                    ModelComponent (Model& owner, const String& name = "",
                                    bool showInTrace = true);
                    ModelComponent (const ModelComponent&);
        virtual    ~ModelComponent ();

                Model&          GetModel() const;   // -> Owner
        const   ExperimentOpts& GetExperimentOpts() const;
```

```
                bool    IsExperimentCompatible (const ModelComponent&) const;
                bool    IsModelCompatible      (const ModelComponent&) const;

        static  Event&          NullEvent ();
                    /*  liefert ein Pseudo-Ereignis, falls auf Anfrage kein
                        entsprechendes gefunden wurde */

        static  Entity&         NullEntity ();
                    /*  liefert ein Pseudo-Entity, falls auf Anfrage kein
                        entsprechendes gefunden wurde */

        static  Process&        NullProcess ();
                    /*  liefert einen Pseudo-Prozess, falls auf Anfrage kein
                        entsprechender gefunden wurde */

        static  SimTime         NOW ();
                    /*  liefert eine Simulationszeit, die ein unmittelbares
                        Eintragen in die Ereignisliste bewirkt */

                SimTime         Epsilon() const;

                SimTime         CurrentTime() const;

                Schedulable&    Current() const;
                    /*  liefert das aktuelle Ereignis oder den
                        gerade laufenden Prozess. Ist weder ein
                        Ereignis noch ein Prozess ermittelbar,
                        wird NullEntity geliefert. */

                Event&          CurrentEvent () const;
                    /*  liefert das aktuelle Ereignis */

                Entity&         CurrentEntity () const;
                    /*  liefert das aktuelle Entity oder NullEntity, falls
                        gerade ein externes Ereignis bearbeitet wird. */

                Process&        CurrentProcess () const;
                    /*  liefert den gerade aktiven Prozess oder NullProcess,
                        falls kein Prozess aktiv ist. */

                Model&          CurrentModel () const;
                    /*  liefert das gerade simulierte Modell. */

                bool            ShowInTrace () const;
                    /*  true, wenn skipTraceNotes <= 0,
                        dekrementiert skipTraceNotes */

                void            ShowInTrace (bool show);
                    /*  Schaltet Trace-Ausgaben fuer diese
                        Komponente ein bzw. aus */

                void            SkipTraceNote (unsigned skip = 1) const;
                    /*  Erhoeht die Trace-Ausgabe-Unterdrueckung
                        um 'skip' Notizen */

//              bool            Valid () const;
                    /*  erweitert NamedObject::Valid() um Abfrage auf
                        Null-Objekte, die als ungueltig zaehlen */

        virtual String          Debug () const;
                                    /*  Gibt Debug-Informationen zurueck */
                String          ClassName () const;

        static  ostream&        Out();
        static  ostream&        Err();
        static  istream&        In();

    protected:
                bool            TraceIsOn() const;
                                    /*  liefert true, wenn der Trace ein-
                                        geschaltet und 'neverTrace'
                                        nicht gesetzt ist */
                void            TraceOn();
                                    /*  schaltet den Trace ein */
                void            TraceOff();
                                    /*  schaltet den Trace aus */
                bool            DebugIsOn() const;
                                    /*  liefert true, wenn der Debug ein-
                                        geschaltet ist und 'neverTrace'
                                        nicht gesetzt ist */
                void            DebugOn();
                                    /*  schaltet die Debug-Funktion ein */
                void            DebugOff();
                                    /*  schaltet die Debug-Funktion aus */
```

```
                void        Warning    (const String& what,
                                        const String& where,
                                        const String& Consequences = "",
                                        const String& hint = "") const;
                void        Error      (const String& what,
                                        const String& where,
                                        const String& Consequences = "",
                                        const String& hint = "") const;
                void        FatalError (const String& what,
                                        const String& where,
                                        const String& Consequences = "",
                                        const String& hint = "") const;
                void        TraceNote (const String& what) const;

                void        SendMessage (const Message&) const;

                bool        valid (const char* className, const char* where,
                                   Message::CodeType    treatAs =
                                   Message::normalError) const;
                    /*  gibt globale Warnung aus, wenn nicht gueltig,
                        className ist die Klasse von this */

                bool        valid ( const ModelComponent& m,
                                    const char* className,
                                    const char* where) const;
                    /*  gibt Warnung aus, wenn m nicht gueltig
                        what ist die Klasse von m
                        setzt voraus, dass this gueltig ist */
    private:
                ModelComponent& operator= (const ModelComponent&);
                                    // Zuweisung nicht implementiert

        Model&          ownerModel;
        bool            showInTrace;    /*  false => Trace- und Debug-
                                            Ausgaben ueber dieses Objekt
                                            werden unterdrueckt */

        // NullObjekte, werden vom ExperimentManager angelegt
        static  Event*      nullEvent;
        static  Process*    nullProcess;
};

// ----------------------------------------------------------------------------

#endif  // MODELCOMPONENT_H
```

modelcom.cc

```
// ----------------------------------------------------------------------------
//
// Datei
//          modelcom.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ----------------------------------------------------------------------------

#include "modelcom.h"

#include "emessage.h"
#include "entity.h"
#include "event.h"
#include "experime.h"
#include "experimm.h"
#include "model.h"
#include "messagem.h"
#include "process.h"
```

```
#include <assert.h>

// --------------------------------------------------------------------------

static const char* className = "ModelComponent";

// --------------------------------------------------------------------------

Event*      ModelComponent::nullEvent   = ExperimentManager::Instance().
                                              GetNullEvent();
Process*    ModelComponent::nullProcess = ExperimentManager::Instance().
                                              GetNullProcess();

// --------------------------------------------------------------------------
// Klassen-Methoden:

Event& ModelComponent::NullEvent ()
    {
        assert (nullEvent);
        return *nullEvent;
     }

// --------------------------------------------------------------------------

Entity& ModelComponent::NullEntity ()
    {
        assert (nullProcess);
        return *nullProcess;
     }

// --------------------------------------------------------------------------

Process& ModelComponent::NullProcess ()
    {
        assert (nullProcess);
        return *nullProcess;
     }

// --------------------------------------------------------------------------

SimTime ModelComponent::NOW ()
    {
        return SimTime::Now();
    }

// --------------------------------------------------------------------------

ModelComponent::ModelComponent (Model& owner, const String& name, bool tr)
    :   NamedObject     (name),
        ownerModel      (owner),
        showInTrace     (tr)
    {}

// --------------------------------------------------------------------------

ModelComponent::ModelComponent (const ModelComponent& mc)
    :   NamedObject     (mc),
        ownerModel      (mc.ownerModel),
        showInTrace     (mc.showInTrace)
    {}

// --------------------------------------------------------------------------

ModelComponent::~ModelComponent ()
    {}

// --------------------------------------------------------------------------

Model& ModelComponent::GetModel() const
    {   return ownerModel; }

// --------------------------------------------------------------------------

const ExperimentOpts& ModelComponent::GetExperimentOpts() const
    {
        assert (GetModel().Connected());
        return GetModel().GetExperiment().GetOpts();
    }

// --------------------------------------------------------------------------

class MsgGlWrnInvalidObject : public GlobalErrorMessage
{
```

```
    public:
        MsgGlWrnInvalidObject (const String& where, const String& clName,
                               Message::CodeType ct = Message::normalError)
            :   GlobalErrorMessage  (where, ct),
                className           (clName)
            {}
        virtual String  Description() const
            {
                return String("invalid '") + className + "'";
            }
    private:
        String  className;
};

// ------------------------------------------------------------------------------

class MsgWrnInvalidObject : public WarningMessage
{
    public:
        MsgWrnInvalidObject (const String& where, const String& clName)
            :   WarningMessage  (where),
                className           (clName)
            {}
        virtual String  Description() const
            {
                return String("invalid '") + className + "'";
            }
    private:
        String  className;
};

// ------------------------------------------------------------------------------

String ModelComponent::Debug () const
    {
        return "";
    }

// ------------------------------------------------------------------------------

String ModelComponent::ClassName () const
    {
        return className;
    }

// ------------------------------------------------------------------------------

bool ModelComponent::valid (const char* clName,
                            const char* where,
                            Message::CodeType treatAs) const
    {
        if (Valid())
            return true;
        ExperimentManager::Instance().Note (
            MsgGlWrnInvalidObject (where, clName, treatAs));
        return false;
    }

// ------------------------------------------------------------------------------

bool ModelComponent::valid (const ModelComponent& m,
                            const char* clName,
                            const char* where) const
    {
        if (m.Valid())
            return true;
        SendMessage (MsgWrnInvalidObject (where, clName));
        return false;
    }

// ------------------------------------------------------------------------------

bool ModelComponent::IsExperimentCompatible (const ModelComponent& mc) const
    {
        if (&GetModel().GetExperiment() == &mc.GetModel().GetExperiment())
            return true;
        return false;
    }

// ------------------------------------------------------------------------------

bool ModelComponent::IsModelCompatible (const ModelComponent& mc) const
    {
        return GetModel().CheckCompatibility (this, &mc);
```

```
    }

// ---------------------------------------------------------------------------

SimTime ModelComponent::Epsilon() const
    {
        return GetExperimentOpts().Epsilon();
    }

// ---------------------------------------------------------------------------

bool ModelComponent::ShowInTrace () const
    {
        if (showInTrace)
            if (this == &GetModel())
                // Hauptmodell: Rekursion beenden
                return true;
            else
                return GetModel().Connected() && GetModel().ShowInTrace();

        return false;
    }

// ---------------------------------------------------------------------------

void ModelComponent::ShowInTrace (bool show)
    {
        showInTrace = show;
    }

// ---------------------------------------------------------------------------

void ModelComponent::SkipTraceNote (unsigned skip) const
    {
        if (ShowInTrace())
            ExperimentManager::Instance().GetMessageManager (*this).
                Skip (skip, Message::trace);
    }

// ---------------------------------------------------------------------------

SimTime ModelComponent::CurrentTime() const
    {
        return ExperimentManager::Instance().CurrentTime (*this);
    }

// ---------------------------------------------------------------------------

Schedulable& ModelComponent::Current () const
    {
        return ExperimentManager::Instance().Current (*this);
    }

// ---------------------------------------------------------------------------

Event& ModelComponent::CurrentEvent () const
    {
        return ExperimentManager::Instance().CurrentEvent (*this);
    }

// ---------------------------------------------------------------------------

Entity& ModelComponent::CurrentEntity () const
    {
        return ExperimentManager::Instance().CurrentEntity (*this);
    }

// ---------------------------------------------------------------------------

Process& ModelComponent::CurrentProcess () const
    {
        return ExperimentManager::Instance().CurrentProcess (*this);
    }

// ---------------------------------------------------------------------------

Model& ModelComponent::CurrentModel () const
    {
        return ExperimentManager::Instance().CurrentModel (*this);
    }

// ---------------------------------------------------------------------------

bool ModelComponent::TraceIsOn() const
```

```
    {
        if (ShowInTrace() && GetModel().Connected())
            return GetModel().GetExperiment().TraceIsOn();
        else
            return false;
    }

// ----------------------------------------------------------------------------

void ModelComponent::TraceOn()
    {
        if (GetModel().Connected())
            GetModel().GetExperiment().TraceOn();
    }

// ----------------------------------------------------------------------------

void ModelComponent::TraceOff()
    {
        if (GetModel().Connected())
            GetModel().GetExperiment().TraceOff();
    }

// ----------------------------------------------------------------------------

bool ModelComponent::DebugIsOn() const
    {
        if (GetModel().Connected())
            return GetModel().GetExperiment().DebugIsOn();
        else
            return false;
    }

// ----------------------------------------------------------------------------

void ModelComponent::DebugOn()
    {
        if (GetModel().Connected())
            GetModel().GetExperiment().DebugOn();
    }

// ----------------------------------------------------------------------------

void ModelComponent::DebugOff()
    {
        if (GetModel().Connected())
            GetModel().GetExperiment().DebugOff();
    }

// ----------------------------------------------------------------------------

void ModelComponent::Warning (const String& what, const String& where,
                              const String& consequences,
                              const String& hint) const
    {
        SendMessage (CustomErrorMessage
                        (where, what, consequences, hint, Message::warning));
    }

// ----------------------------------------------------------------------------

void ModelComponent::Error (const String& what, const String& where,
                            const String& consequences,
                            const String& hint) const
    {
        SendMessage (CustomErrorMessage(where, what, consequences, hint));
    }

// ----------------------------------------------------------------------------

void ModelComponent::FatalError (const String& what, const String& where,
                                 const String& consequences,
                                 const String& hint) const
    {
        SendMessage (CustomErrorMessage
                     (where, what, consequences, hint, Message::fatalError));
    }

// ----------------------------------------------------------------------------

void ModelComponent::TraceNote (const String& what) const
    {
        if (TraceIsOn())
            SendMessage (CustomTraceMessage(what));
```

```
    }

// ---------------------------------------------------------------------------

void ModelComponent::SendMessage (const Message& msg) const
    {
        ExperimentManager::Instance().Note (msg, *this);
    }

// ---------------------------------------------------------------------------

ostream& ModelComponent::Out()
{
    return ExperimentManager::Instance().Out();
}

// ---------------------------------------------------------------------------

ostream& ModelComponent::Err()
{
    return ExperimentManager::Instance().Err();
}

// ---------------------------------------------------------------------------

istream& ModelComponent::In()
{
    return ExperimentManager::Instance().In();
}

// ---------------------------------------------------------------------------
```

msgcomp.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          msgcomp.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#ifndef MSGCOMPONENTS_H
#define MSGCOMPONENTS_H

/*  Allgemeine Konsequenz- und Hinweistexte
*/

#include "str.h"

// ---------------------------------------------------------------------------
// Standardkonsequenzen
// ---------------------------------------------------------------------------

class MsgConsequence
{       //  Programm abbrechen
    public:
        virtual String  Consequences () const;
};

// ---------------------------------------------------------------------------

class MsgCnsqCustom : public MsgConsequence
{       //  Programm abbrechen
    public:
        MsgCnsqCustom (const String& consequence);
        virtual String  Consequences () const;
```

```
    private:
        const   String  consequence;
};

// ------------------------------------------------------------------------------

class MsgCnsqAbortPrg : public MsgConsequence
{       //  Programm abbrechen
    public:
        virtual String  Consequences () const;
};

// ------------------------------------------------------------------------------

class MsgCnsqTerminatePrg : public MsgConsequence
{       //  Programm beenden
    public:
        virtual String  Consequences () const;
};

// ------------------------------------------------------------------------------

class MsgCnsqIgnoreCall : public MsgConsequence
{       //  Methodenaufruf ignorieren
    public:
                MsgCnsqIgnoreCall (const String& call);
        virtual String  Consequences () const;
    protected:
        const   String  call;
};

// ------------------------------------------------------------------------------
// Standardhinweise
// ------------------------------------------------------------------------------

class MsgHint
{       //  Programm abbrechen
    public:
        virtual String  Hint () const;
};

// ------------------------------------------------------------------------------

class MsgHintCustom : public MsgHint
{       //  Programm abbrechen
    public:
        MsgHintCustom (const String& hint);
        virtual String  Hint () const;
    private:
        const   String  hint;
};

// ------------------------------------------------------------------------------

#endif  // MSGCOMPONENTS_H
```

msgcomp.cc

```
// ------------------------------------------------------------------------------
//
// Datei
//         msgcomp.cc
//
// Diplomarbeit
//
//         DESMO-C
//         Implementierung eines Simulators fuer
//         zeitdiskrete Simulation in C++
//
// Autor
//         Thomas Schniewind
//
// Datum
//         8.3.1998
//
// ------------------------------------------------------------------------------

#include "msgcomp.h"
```

```
// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

String MsgConsequence::Consequences () const
    {
        return "";
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

MsgCnsqCustom::MsgCnsqCustom (const String& cons)
    :   consequence (cons)
    {}

// ---------------------------------------------------------------------------

String MsgCnsqCustom::Consequences () const
    {
        return consequence;
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

String MsgCnsqAbortPrg::Consequences () const
    {
        return "The program must be aborted";
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

String MsgCnsqTerminatePrg::Consequences () const
    {
        return "The program must be terminated";
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

MsgCnsqIgnoreCall::MsgCnsqIgnoreCall (const String& Call)
    :   call (Call)
    {}

// ---------------------------------------------------------------------------

String MsgCnsqIgnoreCall::Consequences () const
    {
        return "The call of " + call + " is beeing ignored";
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

String MsgHint::Hint () const
    {
        return "";
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

MsgHintCustom::MsgHintCustom (const String& h)
    :   hint (h)
    {}

// ---------------------------------------------------------------------------

String MsgHintCustom::Hint () const
    {
        return hint;
    }

// ---------------------------------------------------------------------------
```

msgcondq.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          msgcondq.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#ifndef MSGCONDQ_H
#define MSGCONDQ_H

// ---------------------------------------------------------------------------

#include "msgtypes.h"   // Basisklasse

// ---------------------------------------------------------------------------

class CondQueue;
class Condition;

// ---------------------------------------------------------------------------

class TrcCQ : public TraceMessage
{
    protected:
                    TrcCQ (const CondQueue& cq);

        const   CondQueue&  condQ;
};

// ---------------------------------------------------------------------------

class TrcCQWaitUntil : public TrcCQ
{
    public:
                    TrcCQWaitUntil (const CondQueue& cq, const Condition& c);

        virtual String  Description () const;
    protected:
        const   Condition&  cond;
};

// ---------------------------------------------------------------------------

class TrcCQSignal : public TrcCQ
{
    public:
                    TrcCQSignal (const CondQueue& cq);

        virtual String  Description () const;
};

// ---------------------------------------------------------------------------

class TrcCQLeave : public TrcCQ
{
    public:
                    TrcCQLeave (const CondQueue& cq, const Condition& c);

        virtual String  Description () const;
    protected:
        const   Condition&  cond;
};

// ---------------------------------------------------------------------------

#endif  // MSGCONDQ_H
```

msgcondq.cc

```
// ---------------------------------------------------------------------------
//
// Datei
//          msgcondq.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#include "msgcondq.h"

#include "condq.h"
#include "conditio.h"

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

TrcCQ::TrcCQ (const CondQueue& condq)
    :   TraceMessage    (),
        condQ           (condq)
    {}

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

TrcCQWaitUntil::TrcCQWaitUntil (const CondQueue& condQ, const Condition& c)
    :   TrcCQ   (condQ),
        cond    (c)
    {}

// ---------------------------------------------------------------------------

String TrcCQWaitUntil::Description () const
    {
        if (cond.Name() == "")
            return "waits in " + condQ.QuotedName() + " until ...";
        else
            return  "waits in " + condQ.QuotedName() + " until "
                    + cond.QuotedName();
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

TrcCQSignal::TrcCQSignal (const CondQueue& condQ)
    :   TrcCQ   (condQ)
    {}

// ---------------------------------------------------------------------------

String TrcCQSignal::Description () const
    {
        return "signals " + condQ.QuotedName();
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

TrcCQLeave::TrcCQLeave (const CondQueue& condQ, const Condition& c)
    :   TrcCQ   (condQ),
        cond    (c)
    {}

// ---------------------------------------------------------------------------

String TrcCQLeave::Description () const
    {
        if (cond.Name() == "")
            return "leaves " + condQ.QuotedName();
        else
```

```
            return  "leaves " + condQ.QuotedName() + ", cause "
                    + cond.QuotedName();
    }

// ---------------------------------------------------------------------------
```

msgdist.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          msgdist.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#ifndef DISTMESSAGES_H
#define DISTMESSAGES_H

// ---------------------------------------------------------------------------

#include "msgtypes.h"   // Basisklasse

#include "msgcomp.h"
#include "str.h"

// ---------------------------------------------------------------------------

class Distribution;

// ---------------------------------------------------------------------------
// allgemeine Mixin-Klasse fuer alle Meldungen einer Distribution

class MsgDist
{
    protected:
                MsgDist (const Distribution& d);

        const   Distribution&   dist;
};

// ---------------------------------------------------------------------------
// Warnungen
// ---------------------------------------------------------------------------
// Kopieren eines bereits benutzten ZZ-Stroms

class MsgDistChangeUsed : public WarningMessage, public MsgDist
{
    public:
                MsgDistChangeUsed ( const String& where,
                                    const Distribution& d);
        virtual String  Description() const;
};

// ---------------------------------------------------------------------------
// Kopieren eines bereits benutzten ZZ-Stroms

class MsgDistCopyUsed : public WarningMessage, public MsgDist
{
    public:
                MsgDistCopyUsed (   const String& where,
                                    const Distribution& d);
        virtual String  Description() const;
};

// ---------------------------------------------------------------------------
// Ungueltiger Eintragindex bei Empirical
```

```
class MsgDistEmpWrongIndex : public WarningMessage, public MsgDist
{
    public:
            MsgDistEmpWrongIndex (  const String& where,
                                    const Distribution& d,
                                    const unsigned index,
                                    const unsigned bound);
        virtual String  Description() const;
    protected:
        const   unsigned    index, bound;
};

// -------------------------------------------------------------------------
// Erlang-Verteilung mit k = 0

class MsgDistErlZeroK : public WarningMessage, public MsgDist
{
    public:
            MsgDistErlZeroK (   const String&       where,
                                const Distribution& d);
        virtual String  Description() const;
};

// -------------------------------------------------------------------------
// ZZ-Strom mit negativem Mittelwert

class MsgDistMeanNeg : public WarningMessage, public MsgDist
{
    public:
            MsgDistMeanNeg (const String&       where,
                            const Distribution& d,
                            const double        mean);
        virtual String  Description() const;
    protected:
        const   double  mean;
};

// -------------------------------------------------------------------------
// ZZ-Strom mit negativer Wahrscheinlichkeit

class MsgDistProbNeg : public WarningMessage, public MsgDist
{
    public:
            MsgDistProbNeg (const String& where,
                            const Distribution& d,
                            const double prob);
        virtual String  Description() const;
    protected:
        const   double  prob;
};

// -------------------------------------------------------------------------
// ZZ-Strom mit zu grosser Wahrscheinlichkeit

class MsgDistProbTooBig : public WarningMessage, public MsgDist
{
    public:
            MsgDistProbTooBig ( const String& where,
                                const Distribution& d,
                                const double prob);
        virtual String  Description() const;
    protected:
        const   double  prob;
};

// -------------------------------------------------------------------------
// Normal-Verteilung mit negativer Standardabweichung

class MsgDistStdDevNeg : public WarningMessage, public MsgDist
{
    public:
            MsgDistStdDevNeg (  const String&       where,
                                const Distribution& d,
                                const double        stddev);
        virtual String  Description() const;
    protected:
        const   double  stddev;
};

// -------------------------------------------------------------------------
// Vertauschte Grenzen bei Uniform

class MsgDistUnifSwap : public WarningMessage, public MsgDist
```

```
{
    public:
                MsgDistUnifSwap (   const String& where,
                                    const Distribution& d,
                                    const int low,
                                    const int high);
                MsgDistUnifSwap (   const String& where,
                                    const Distribution& d,
                                    const double low,
                                    const double high);
        virtual String  Description() const;
};

// ------------------------------------------------------------------------
// Fehler
// ------------------------------------------------------------------------
// Nicht passendes Parameterpaar fuer empirische Verteilungen

class MsgDistEmpProbOrder : public ErrorMessage, public MsgDist
{
    public:
                MsgDistEmpProbOrder (   const String&       where,
                                        const Distribution& d,
                                        const double        prob1,
                                        const double        prob2);
        virtual String  Description() const;
    protected:
        const   double  p1, p2;
};

// ------------------------------------------------------------------------
// Benutzen eines nicht initialisierten ZZ-Stroms

class MsgDistNoInit : public ErrorMessage, public MsgDist
{
    public:
                MsgDistNoInit ( const String& where,
                                const Distribution& d);
        virtual String  Description() const;
};

// ------------------------------------------------------------------------
// Fatale Fehler
// ------------------------------------------------------------------------
// Kopieren eines bereits geloeschten ZZ-Stroms

class MsgDistCopyInvalid : public FatalErrorMessage, public MsgDist
{
    public:
                MsgDistCopyInvalid (    const String& where,
                                        const Distribution& d);
        virtual String  Description() const;
};

// ------------------------------------------------------------------------
// Benutzen eines bereits geloeschten ZZ-Stroms

class MsgDistUseDeleted : public FatalErrorMessage
{
    public:
                MsgDistUseDeleted ( const String& where);
        virtual String  Description() const;
};

// ------------------------------------------------------------------------
// allgemeine Konsequenz-Klassen fuer Meldungen einer Distribution
// ------------------------------------------------------------------------

class MsgDistCnsqResetCopy : public MsgConsequence
{
    public:
        virtual String  Consequences () const;
};

// ------------------------------------------------------------------------

class MsgDistCnsqRetUndef : public MsgConsequence
{
    public:
        virtual String  Consequences () const;
};

// ------------------------------------------------------------------------
```

```
class MsgDistCnsqRetHighValue : public MsgConsequence
{
    public:
        virtual String  Consequences () const;
};

// ---------------------------------------------------------------------------

class MsgDistCnsqISwapped : public MsgConsequence
{
    public:
                MsgDistCnsqISwapped (int low, int high);
        virtual String  Consequences () const;
    protected:
        const   int     low;
        const   int     high;
};

// ---------------------------------------------------------------------------

class MsgDistCnsqDSwapped : public MsgConsequence
{
    public:
                MsgDistCnsqDSwapped (double low, double high);
        virtual String  Consequences () const;
    protected:
        const   double  low;
        const   double  high;
};

// ---------------------------------------------------------------------------

class MsgDistCnsqUseAbsolute : public MsgConsequence
{
    public:
        virtual String  Consequences () const;
};

// ---------------------------------------------------------------------------

class MsgDistCnsqUseOne : public MsgConsequence
{
    public:
        virtual String  Consequences () const;
};

// ---------------------------------------------------------------------------

class MsgDistCnsqUseZero : public MsgConsequence
{
    public:
        virtual String  Consequences () const;
};

// ---------------------------------------------------------------------------
// allgemeine Hinweis-Klassen fuer Meldungen einer Distribution
// ---------------------------------------------------------------------------

class MsgDistHintChangeUsed : public MsgHint
{
    public:
        virtual String  Hint () const;
};

// ---------------------------------------------------------------------------
// Trace-Meldungen
// ---------------------------------------------------------------------------
// BoolDist::Sample()

class TrcDistBSample : public TraceMessage, public MsgDist
{
    public:
                TrcDistBSample (    const   Distribution&   d,
                                        bool            b);
        virtual String  Description() const;
    protected:
        const   bool    b;
};

// ---------------------------------------------------------------------------
// IntDist::Sample()

class TrcDistISample : public TraceMessage, public MsgDist
{
```

```
    public:
                TrcDistISample (    const   Distribution&   d,
                                        int             i);
        virtual String  Description() const;
    protected:
        const   int     i;
};

// ------------------------------------------------------------------------------
// RealDist::Sample()

class TrcDistRSample : public TraceMessage, public MsgDist
{
    public:
                TrcDistRSample (    const   Distribution&   d,
                                        double          r);
        virtual String  Description() const;
    protected:
        const   double  r;
};

// ------------------------------------------------------------------------------

#endif  // DISTMESSAGES_H
```

msgdist.cc

```
// ------------------------------------------------------------------------------
//
// Datei
//          msgdist.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ------------------------------------------------------------------------------

#include "msgdist.h"

#include "distribu.h"
#include <iomanip.h>

// ------------------------------------------------------------------------------

MsgDist::MsgDist (const Distribution& d)
    :   dist(d)
    {}

// ------------------------------------------------------------------------------
// Warnungen

MsgDistCopyInvalid::MsgDistCopyInvalid (    const String& Where,
                                            const Distribution& d)
    :   FatalErrorMessage   (Where),
        MsgDist             (d)
    {}

String MsgDistCopyInvalid::Description () const
    {
        strstream ss;
        ss << "Attempt to copy an invalid distribution" << ends;
        return ss;
    }

// ------------------------------------------------------------------------------

MsgDistCopyUsed::MsgDistCopyUsed (  const String& Where,
                                    const Distribution& d)
    :   WarningMessage  (Where, new MsgDistCnsqResetCopy()),
```

```
        MsgDist         (d)
    {}

String MsgDistCopyUsed::Description () const
    {
        strstream ss;
        ss  << "Attempt to copy " << Quote (dist.Name())
            << " that has already been used" << ends;
        return ss;
    }

// ---------------------------------------------------------------------------

MsgDistChangeUsed::MsgDistChangeUsed (  const String& Where,
                                        const Distribution& d)
    :   WarningMessage  (Where, new MsgConsequence(),
                                new MsgDistHintChangeUsed()),
        MsgDist         (d)
    {}

String MsgDistChangeUsed::Description () const
    {
        strstream ss;
        ss  << "Attempt to change parameters of distribution "
            << Quote (dist.Name())
            << " that has already been used" << ends;
        return ss;
    }

// ---------------------------------------------------------------------------

MsgDistProbNeg::MsgDistProbNeg (    const String&       Where,
                                        const Distribution& d,
                                        const double        p)
    :   WarningMessage  (Where, new MsgDistCnsqUseZero()),
        MsgDist         (d),
        prob            (p)
    {}

String MsgDistProbNeg::Description () const
    {
        strstream ss;
        ss  << "Negative probability value " << prob
            << " for distribution " << Quote (dist.Name()) << ends;
        return ss;
    }

// ---------------------------------------------------------------------------

MsgDistProbTooBig::MsgDistProbTooBig (  const String&       Where,
                                        const Distribution& d,
                                        const double        p)
    :   WarningMessage  (Where, new MsgDistCnsqUseOne()),
        MsgDist         (d),
        prob            (p)
    {}

String MsgDistProbTooBig::Description () const
    {
        strstream ss;
        ss  << "Probability value " << prob << " greater than 1.0"
            << " for distribution " << Quote (dist.Name()) << ends;
        return ss;
    }

// ---------------------------------------------------------------------------

MsgDistUnifSwap::MsgDistUnifSwap (  const String&       Where,
                                    const Distribution& d,
                                    const int           low,
                                    const int           high)
    :   WarningMessage  (Where, new MsgDistCnsqISwapped(low, high)),
        MsgDist         (d)
    {}

MsgDistUnifSwap::MsgDistUnifSwap (  const String&       Where,
                                    const Distribution& d,
                                    const double        low,
                                    const double        high)
    :   WarningMessage  (Where, new MsgDistCnsqDSwapped(low, high)),
        MsgDist         (d)
    {}

String MsgDistUnifSwap::Description () const
```

```
    {
        return  "Upper bound less than lower bound for distribution " +
                Quote (dist.Name()) + ". Bounds swapped";
    }

// --------------------------------------------------------------------------

MsgDistEmpWrongIndex::MsgDistEmpWrongIndex (    const String&       Where,
                                                const Distribution& d,
                                                const unsigned      Index,
                                                const unsigned      Bound)
    :   WarningMessage  (Where, new MsgDistCnsqRetHighValue()),
        MsgDist         (d),
        index           (Index),
        bound           (Bound)
    {}

String MsgDistEmpWrongIndex::Description () const
    {
        return  "Can't get entry " + String(index) + " of distribution " +
                Quote (dist.Name()) + " because it has only " + bound +
                " entries";
    }

// --------------------------------------------------------------------------

MsgDistMeanNeg::MsgDistMeanNeg (const String&       Where,
                                const Distribution& d,
                                const double        m)
    :   WarningMessage  (Where, new MsgDistCnsqUseAbsolute()),
        MsgDist         (d),
        mean            (m)
    {}

String MsgDistMeanNeg::Description () const
    {
        return  "Negative mean value " + String(mean) +
                " for distribution " + Quote (dist.Name());
    }

// --------------------------------------------------------------------------

MsgDistErlZeroK::MsgDistErlZeroK (  const String&       Where,
                                    const Distribution& d)
    :   WarningMessage  (Where, new MsgDistCnsqUseOne()),
        MsgDist         (d)
    {}

String MsgDistErlZeroK::Description () const
    {
        return  "k-Erlang distribution " + Quote (dist.Name()) +
                " specified with k = 0.";
    }

// --------------------------------------------------------------------------

MsgDistStdDevNeg::MsgDistStdDevNeg (const String&       Where,
                                    const Distribution& d,
                                    const double        Stddev)
    :   WarningMessage  (Where, new MsgDistCnsqUseAbsolute()),
        MsgDist         (d),
        stddev          (Stddev)
    {}

String MsgDistStdDevNeg::Description () const
    {
        return  "Negative value " + String(stddev) + "for standard deviance "
        "of distribution " + Quote (dist.Name());
    }

// --------------------------------------------------------------------------
// Fehler

MsgDistNoInit::MsgDistNoInit (  const String& Where,
                                const Distribution& d)
    :   ErrorMessage    (Where),
        MsgDist         (d)
    {}

String MsgDistNoInit::Description () const
    {
        strstream ss;
        ss  << "Attempt to call method " << Quote (where)
            << " of uninitialized distribution " << Quote (dist.Name())
```

```
            << ends;
        return ss;
    }

// ---------------------------------------------------------------------------

MsgDistEmpProbOrder::MsgDistEmpProbOrder (  const String&       Where,
                                            const Distribution& d,
                                            const double        prob1,
                                            const double        prob2)
    :   ErrorMessage    (Where),
        MsgDist         (d),
        p1              (prob1),
        p2              (prob2)
    {}

String MsgDistEmpProbOrder::Description () const
    {
        return  "Can't add probability (" + String(p1) + ") to distribution "
                + Quote (dist.Name()) + " because it is less than a"
                "previous one (" + String(p2) + ")";
    }

// ---------------------------------------------------------------------------
// Fatale Fehler

MsgDistUseDeleted::MsgDistUseDeleted (  const String& Where)
    :   FatalErrorMessage   (Where)
    {}

String MsgDistUseDeleted::Description () const
    {
        strstream ss;
        ss  << "Attempt to call method " << Quote (where)
            << " of an already deleted distribution" << ends;
        return ss;
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

String MsgDistCnsqResetCopy::Consequences() const
    {
        return "A reset copy is beeing constructed";
    }

// ---------------------------------------------------------------------------

String MsgDistCnsqRetUndef::Consequences() const
    {
        return "'Undefined' is beeing returned";
    }

// ---------------------------------------------------------------------------

String MsgDistCnsqRetHighValue::Consequences() const
    {
        return "the highest value is beeing returned";
    }

// ---------------------------------------------------------------------------

String MsgDistCnsqUseZero::Consequences() const
    {
        return "0.0 used";
    }

// ---------------------------------------------------------------------------

String MsgDistCnsqUseOne::Consequences() const
    {
        return "1.0 used";
    }

// ---------------------------------------------------------------------------

String MsgDistCnsqUseAbsolute::Consequences() const
    {
        return "Absolute value used";
    }

// ---------------------------------------------------------------------------

MsgDistCnsqISwapped::MsgDistCnsqISwapped (int l, int h)
```

```
    :   low (l), high (h)
    {}

String MsgDistCnsqISwapped::Consequences() const
    {
        strstream   ss;
        ss  << "Lower bound is now " << low
            << ", upper bound is " << high << ends;
        return ss;
    }

// ---------------------------------------------------------------------------

MsgDistCnsqDSwapped::MsgDistCnsqDSwapped (double l, double h)
    :   low (l), high (h)
    {}

String MsgDistCnsqDSwapped::Consequences() const
    {
        strstream   ss;
        ss  << "Lower bound is now "
            << setiosflags (ios::fixed) << setprecision(3) << low
            << ", upper bound is "
            << setiosflags (ios::fixed) << setprecision(3) << high << ends;
        return ss;
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

String MsgDistHintChangeUsed::Hint() const
    {
        return
            "You are not allowed to change parameters of a distribution "
            "after calls to method 'Sample()'. If you need a different "
            "distribution, use a second distribution object.";
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

TrcDistBSample::TrcDistBSample (const Distribution& d, bool B)
    :   TraceMessage(),
        MsgDist     (d),
        b           (B)
    {}

String TrcDistBSample::Description() const
    {
        return  "samples " + Quote (String(b)) + " from " +
                Quote (dist.Name());
    }
// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

TrcDistISample::TrcDistISample (const Distribution& d, int I)
    :   TraceMessage(),
        MsgDist     (d),
        i           (I)
    {}

String TrcDistISample::Description() const
    {
        return  "samples " + String(i) + " from " +
                Quote (dist.Name());
    }


// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

TrcDistRSample::TrcDistRSample (const Distribution& d, double R)
    :   TraceMessage(),
        MsgDist     (d),
        r           (R)
    {}

String TrcDistRSample::Description() const
    {
        return  "samples " + String(r) + " from " +
                Quote (dist.Name());
    }

// ---------------------------------------------------------------------------
```

msgqueue.h

```
// --------------------------------------------------------------------------
//
// Datei
//          msgqueue.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// --------------------------------------------------------------------------

#ifndef QUEUEMESSAGES_H
#define QUEUEMESSAGES_H

// --------------------------------------------------------------------------

#include "msgtypes.h"   // Basisklassen

#include "str.h"

// --------------------------------------------------------------------------

class QueueBased;
class Condition;

// --------------------------------------------------------------------------
// allgemeine Oberklassen fuer Warnungen einer Queue

class QMsgWarning : public WarningMessage
{
    protected:
                QMsgWarning (const String& where, const QueueBased& q,
                             const Entity& obj1, const Entity& obj2);

        const   QueueBased& queue;
        const   Entity&     obj1;
        const   Entity&     obj2;
};

// --------------------------------------------------------------------------

class QMsgIncompatible : public QMsgWarning
{
    public:
        virtual String  Hint () const;
    protected:
                QMsgIncompatible (const String& where, const QueueBased& q,
                                  const Entity& e);
};

// --------------------------------------------------------------------------
// Insert

class QMsgInsertInvalid : public QMsgWarning
{
    public:
                QMsgInsertInvalid (const String& where, const QueueBased& q);
        virtual String  Description () const;
};

// --------------------------------------------------------------------------

class QMsgInsertNullEntity : public QMsgWarning
{
    public:
                QMsgInsertNullEntity (const String&     where,
                                      const QueueBased& q);
        virtual String  Description () const;
};

// --------------------------------------------------------------------------
```

```
class QMsgInsertIncompatible : public QMsgIncompatible
{
    public:
                QMsgInsertIncompatible (const String&     where,
                                        const QueueBased& q,
                                        const Entity&     e);
        virtual String  Description () const;
};

// -----------------------------------------------------------------------
// InsertAt

class QMsgInsertAt : public QMsgWarning
{
    protected:
                QMsgInsertAt (const String& where, const QueueBased& q,
                              const Entity& obj1,  const Entity&     obj2,
                              const String& at);
        const   String      at; // vor oder nach obj2
};

// -----------------------------------------------------------------------

class QMsgInsertAtInvalid : public QMsgInsertAt
{
    public:
                QMsgInsertAtInvalid (const String& where, const QueueBased& q,
                                     const Entity& e1,    const String& at);
        virtual String  Description () const;
};

// -----------------------------------------------------------------------

class QMsgInsertAtNullEntity : public QMsgInsertAt
{
    public:
                QMsgInsertAtNullEntity (const String& where,
                                        const QueueBased& q,
                                        const Entity& e1, const String& at);
        virtual String  Description () const;
};

// -----------------------------------------------------------------------

class QMsgInsertAtNotFound : public QMsgInsertAt
{
    public:
                QMsgInsertAtNotFound (const String& where,
                                      const QueueBased& q,
                                      const Entity& e1, const Entity& e2,
                                      const String& at);
        virtual String  Description () const;
};

// -----------------------------------------------------------------------

class QMsgInsertAtIncompatible : public QMsgInsertAt
{
    public:
                QMsgInsertAtIncompatible (const String& where,
                                          const QueueBased& q,
                                          const Entity& e1, const Entity& e2,
                                          const String& at);
        virtual String  Description () const;
};

// -----------------------------------------------------------------------
// Remove

class QMsgRemoveInvalid : public QMsgWarning
{
    public:
                QMsgRemoveInvalid (const String& where, const QueueBased& q);
        virtual String  Description () const;
};

// -----------------------------------------------------------------------

class QMsgRemoveNullEntity : public QMsgWarning
{
    public:
                QMsgRemoveNullEntity (const String& where,
                                      const QueueBased& q);
        virtual String  Description () const;
```

```
};

// ---------------------------------------------------------------------------

class QMsgRemoveIncompatible : public QMsgWarning
{
    public:
                QMsgRemoveIncompatible (const String& where,
                                        const QueueBased& q,
                                        const Entity& e);
        virtual String  Description () const;
};

// ---------------------------------------------------------------------------

class QMsgRemoveNotFound : public QMsgWarning
{
    public:
                QMsgRemoveNotFound (const String& where, const QueueBased& q,
                                    const Entity& e);
        virtual String  Description () const;
};

// ---------------------------------------------------------------------------
// Pred/Succ

class QMsgPredSucc : public QMsgWarning
{
    public:
                QMsgPredSucc (const String& where, const QueueBased& q,
                              const Entity& e, const String& PorS);
        virtual String  Consequences () const;
    protected:
        const   String      predOrSucc;
};

// ---------------------------------------------------------------------------

class QMsgPredSuccNullEntity : public QMsgPredSucc
{
    public:
                QMsgPredSuccNullEntity (const String& where,
                                        const QueueBased& q,
                                        const String& PorS);
        virtual String  Description () const;
};

// ---------------------------------------------------------------------------

class QMsgPredSuccInvalid : public QMsgPredSucc
{
    public:
                QMsgPredSuccInvalid (const String& where, const QueueBased& q,
                                     const String& PorS);
        virtual String  Description () const;
};

// ---------------------------------------------------------------------------

class QMsgPredSuccIncompatible : public QMsgPredSucc
{
    public:
                QMsgPredSuccIncompatible (const String& where,
                                          const QueueBased& q,
                                          const Entity& e,
                                          const String& PorS);
        virtual String  Description () const;
};

// ---------------------------------------------------------------------------

class QMsgPredSuccNotFound : public QMsgPredSucc
{
    public:
                QMsgPredSuccNotFound (const String& where,
                                      const QueueBased& q,
                                      const Entity& e, const String& PorS);
        virtual String  Description () const;
};

// ---------------------------------------------------------------------------

class QMsgIncompatibleCondition : public QMsgIncompatible
{
```

```
    public:
                QMsgIncompatibleCondition (const String& where,
                                           const QueueBased& q,
                                           const Condition& c);
        virtual String  Description () const;
        virtual String  Consequences () const;
    protected:
        const   Condition&  condition;
};

// ---------------------------------------------------------------------------
// Trace-Meldungen
// ---------------------------------------------------------------------------

class TrcQMessage : public TraceMessage
{
    protected:
                TrcQMessage (const QueueBased& q, const Entity& obj1,
                               const Entity& obj2);

        const   QueueBased& queue;
        const   Entity& obj1;
        const   Entity& obj2;
};

// ---------------------------------------------------------------------------

class TrcQInsert : public TrcQMessage
{
    public:
                TrcQInsert (const QueueBased& q, const Entity& e);
        virtual String  Description () const;
};

// ---------------------------------------------------------------------------

class TrcQInsertAt : public TrcQMessage
{
    public:
                TrcQInsertAt (  const QueueBased& q, const Entity& which,
                                    const Entity& where, const String& at);
        virtual String  Description () const;
    protected:
        const   String  at;
};

// ---------------------------------------------------------------------------

class TrcQRemove : public TrcQMessage
{
    public:
                TrcQRemove (const QueueBased& q, const Entity& e);
        virtual String  Description () const;
};

// ---------------------------------------------------------------------------

class TrcQFind : public TrcQMessage
{
    public:
                TrcQFind (const QueueBased& q, const Entity& e);
        virtual String  Description () const;
};

// ---------------------------------------------------------------------------

#endif  // QUEUEMESSAGES_H
```

msgqueue.cc

```
// ---------------------------------------------------------------------------
//
// Datei
//          msgqueue.cc
//
// Diplomarbeit
//
//          DESMO-C
```

```
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#include "msgqueue.h"

#include "conditio.h"
#include "entity.h"
#include "msgcomp.h"
#include "qbased.h"
#include "text.h"

// ---------------------------------------------------------------------------
// Warnungen:

// ---------------------------------------------------------------------------

QMsgWarning::QMsgWarning (const String& Where, const QueueBased& q,
                          const Entity& e1, const Entity& e2)
    :   WarningMessage(Where, new MsgCnsqIgnoreCall(Quote (Where))),
        queue(q),
        obj1(e1),
        obj2(e2)
    {}

// ---------------------------------------------------------------------------

QMsgIncompatible::QMsgIncompatible
            (const String& Where, const QueueBased& q, const Entity& e)
    :   QMsgWarning(Where, q, e, ModelComponent::NullEntity())
    {}

String QMsgIncompatible::Hint () const
    {
        strstream ss;
        ss << "use a 'port' to transfer an entity from one model to another"
           << ends;
        return ss;
    }

// ---------------------------------------------------------------------------
// Insert

QMsgInsertInvalid::QMsgInsertInvalid (const String& Where,
                                      const QueueBased& q)
    :   QMsgWarning(Where, q, ModelComponent::NullEntity(),
                              ModelComponent::NullEntity())
    {}

String QMsgInsertInvalid::Description () const
    {
        strstream ss;
        ss  << "attempt to insert an invalid entity into queue "
            << Quote (queue.Name()) << ends;
        return ss;
    }

// ---------------------------------------------------------------------------

QMsgInsertNullEntity::QMsgInsertNullEntity (const String& Where,
                                            const QueueBased& q)
    :   QMsgWarning(Where, q, ModelComponent::NullEntity(),
                              ModelComponent::NullEntity())
    {}

String QMsgInsertNullEntity::Description () const
    {
        strstream ss;
        ss  << "attempt to insert the " << txtNullEntity << " into queue "
            << Quote (queue.Name()) << ends;
        return ss;
    }

// ---------------------------------------------------------------------------

QMsgInsertIncompatible::QMsgInsertIncompatible
            (const String& Where, const QueueBased& q, const Entity& e)
```

```
    :   QMsgIncompatible(Where, q, e)
    {}

String QMsgInsertIncompatible::Description () const
    {
        strstream ss;
        ss  << "attempt to insert the incompatible entity "
            << NameAndModel (obj1) << " into queue "
            << NameAndModel (queue) << ends;
        return ss;
    }

// ------------------------------------------------------------------------------
// InsertAt

QMsgInsertAt::QMsgInsertAt
        (const String& Where, const QueueBased& q, const Entity& e1,
         const Entity& e2, const String& s)
    :   QMsgWarning(Where, q, e1, e2),
        at(s)
    {}

// ------------------------------------------------------------------------------

QMsgInsertAtInvalid::QMsgInsertAtInvalid
        (const String& Where, const QueueBased& q, const Entity& e,
         const String& at)
    :   QMsgInsertAt(Where, q, e, ModelComponent::NullEntity(), at)
    {}

String QMsgInsertAtInvalid::Description () const
    {
        strstream ss;
        ss  << "attempt to insert entity "
            << Quote (obj1.Name())
            << " into queue "<< Quote (queue.Name())
            << at << " an invalid entity" << ends;
        return ss;
    }

// ------------------------------------------------------------------------------

QMsgInsertAtNullEntity::QMsgInsertAtNullEntity
        (const String& Where, const QueueBased& q, const Entity& e,
         const String& at)
    :   QMsgInsertAt(Where, q, e, ModelComponent::NullEntity(), at)
    {}

String QMsgInsertAtNullEntity::Description () const
    {
        strstream ss;
        ss  << "attempt to insert entity "
            << Quote (obj1.Name())
            << " into queue "<< Quote (queue.Name())
            << at << " the " << txtNullEntity << ends;
        return ss;
    }

// ------------------------------------------------------------------------------

QMsgInsertAtNotFound::QMsgInsertAtNotFound
        (const String& Where, const QueueBased& q, const Entity& e1,
         const Entity& e2, const String& at)
    :   QMsgInsertAt(Where, q, e1, e2, at)
    {}

String QMsgInsertAtNotFound::Description () const
    {
        strstream ss;
        ss  << "attempt to insert entity "
            << Quote (obj1.Name())
            << " into queue "<< Quote (queue.Name())
            << at << " entity " << Quote (obj2.Name())
            << ", which could not be found in this queue" << ends;
        return ss;
    }

// ------------------------------------------------------------------------------

QMsgInsertAtIncompatible::QMsgInsertAtIncompatible
        (const String& Where, const QueueBased& q, const Entity& e1,
         const Entity& e2, const String& at)
    :   QMsgInsertAt(Where, q, e1, e2, at)
    {}
```

```
String QMsgInsertAtIncompatible::Description () const
    {
        strstream ss;
        ss  << "attempt to insert entity "
            << Quote (obj1.Name())
            << " into queue "<< NameAndModel (queue)
            << at << " the incompatible entity " << NameAndModel (obj2)
            << ends;
        return ss;
    }

// ---------------------------------------------------------------------------
// Remove

QMsgRemoveInvalid::QMsgRemoveInvalid (const String& Where,
                                      const QueueBased& q)
    :   QMsgWarning(Where, q, ModelComponent::NullEntity(),
                              ModelComponent::NullEntity())
    {}

String QMsgRemoveInvalid::Description () const
    {
        strstream ss;
        ss  << "attempt to remove an invalid entity from queue "
            << Quote (queue.Name()) << ends;
        return ss;
    }

// ---------------------------------------------------------------------------

QMsgRemoveNullEntity::QMsgRemoveNullEntity (const String& Where,
                                            const QueueBased& q)
    :   QMsgWarning(Where, q, ModelComponent::NullEntity(),
                              ModelComponent::NullEntity())
    {}

String QMsgRemoveNullEntity::Description () const
    {
        strstream ss;
        ss  << "attempt to remove the " << txtNullEntity << " from queue "
            << Quote (queue.Name()) << ends;
        return ss;
    }

// ---------------------------------------------------------------------------

QMsgRemoveIncompatible::QMsgRemoveIncompatible
            (const String& Where, const QueueBased& q, const Entity& e)
    :   QMsgWarning(Where, q, e, ModelComponent::NullEntity())
    {}

String QMsgRemoveIncompatible::Description () const
    {
        strstream ss;
        ss  << "attempt to remove the incompatible entity "
            << NameAndModel (obj1) << " from queue "
            << NameAndModel (queue) << ends;
        return ss;
    }

// ---------------------------------------------------------------------------

QMsgRemoveNotFound::QMsgRemoveNotFound
        (const String& Where, const QueueBased& q, const Entity& e)
    :   QMsgWarning(Where, q, e, ModelComponent::NullEntity())
    {}

String QMsgRemoveNotFound::Description () const
    {
        strstream ss;
        ss  << "attempt to remove  "
            << Quote (obj1.Name())
            << ", which could not be found in queue "
            << Quote (queue.Name()) << ends;
        return ss;
    }

// ---------------------------------------------------------------------------
// Pred/Succ

QMsgPredSucc::QMsgPredSucc (const String& Where, const QueueBased& q,
                            const Entity& e, const String& PorS)
    :   QMsgWarning(Where, q, e, ModelComponent::NullEntity()),
```

```
        predOrSucc(PorS)
    {}

String QMsgPredSucc::Consequences () const
    {
        strstream ss;
        ss  << "the " << txtNullEntity << " will be returned" << ends;
        return ss;
    }

// ---------------------------------------------------------------------------

QMsgPredSuccInvalid::QMsgPredSuccInvalid (const String& Where,
                                          const QueueBased& q,
                                          const String& PorS)
    :   QMsgPredSucc(Where, q, ModelComponent::NullEntity(), PorS)
    {}

String QMsgPredSuccInvalid::Description () const
    {
        strstream ss;
        ss  << "attempt to get " << predOrSucc
            << " of an invalid entity within queue "
            << Quote (queue.Name()) << ends;
        return ss;
    }

// ---------------------------------------------------------------------------

QMsgPredSuccNullEntity::QMsgPredSuccNullEntity (const String& Where,
                                                const QueueBased& q,
                                                const String& PorS)
    :   QMsgPredSucc(Where, q, ModelComponent::NullEntity(), PorS)
    {}

String QMsgPredSuccNullEntity::Description () const
    {
        strstream ss;
        ss  << "attempt to get " << predOrSucc
            << " of the " << txtNullEntity << " within queue "
            << Quote (queue.Name()) << ends;
        return ss;
    }

// ---------------------------------------------------------------------------

QMsgPredSuccNotFound::QMsgPredSuccNotFound
        (const String& Where, const QueueBased& q, const Entity& e,
         const String& PorS)
    :   QMsgPredSucc(Where, q, e, PorS)
    {}

String QMsgPredSuccNotFound::Description () const
    {
        strstream ss;
        ss  << "attempt to get " << predOrSucc
            << " of entity " << Quote (obj1.Name())
            << " , which could not be found in queue "
            << Quote (queue.Name()) << ends;
        return ss;
    }

// ---------------------------------------------------------------------------

QMsgPredSuccIncompatible::QMsgPredSuccIncompatible
        (const String& Where, const QueueBased& q, const Entity& e,
         const String& PorS)
    :   QMsgPredSucc(Where, q, e, PorS)
    {}

String QMsgPredSuccIncompatible::Description () const
    {
        strstream ss;
        ss  << "attempt to get " << predOrSucc
            << " of the incompatible entity "
            << NameAndModel (obj1) << " within queue "
            << NameAndModel (queue) << ends;
        return ss;
    }

// ---------------------------------------------------------------------------

QMsgIncompatibleCondition::QMsgIncompatibleCondition
        (const String& Where, const QueueBased& q, const Condition& c)
```

```
    :   QMsgIncompatible(Where, q, ModelComponent::NullEntity()),
        condition(c)
    {}

String QMsgIncompatibleCondition::Description () const
    {
        strstream ss;
        ss  << "attempt to use the incombatible condition "
            << NameAndModel (condition) << " with queue "
            << NameAndModel (queue) << ends;
        return ss;
    }

String QMsgIncompatibleCondition::Consequences () const
    {
        strstream ss;
        ss  << "the " << txtNullEntity << " will be returned";
        return ss;
    }

// ---------------------------------------------------------------------------
// Trace-Messages:
// ---------------------------------------------------------------------------

TrcQMessage::TrcQMessage (const QueueBased& q, const Entity& e1,
                                               const Entity& e2)
    :   TraceMessage(),
        queue(q),
        obj1(e1),
        obj2(e2)
    {}

// ---------------------------------------------------------------------------
TrcQInsert::TrcQInsert (const QueueBased& q, const Entity& e)
    :   TrcQMessage(q, e, ModelComponent::NullEntity())
    {}

String TrcQInsert::Description () const
    {
        strstream ss;
        ss  << txtInserts << ' ' << TxtItselfIfCurrent (obj1)
            << ' ' << txtInto << ' ' << Quote (queue.Name()) << ends;
        return ss;
    }

// ---------------------------------------------------------------------------
TrcQInsertAt::TrcQInsertAt (    const QueueBased& q, const Entity& e,
                                const Entity&     w, const String& Where)
    :   TrcQMessage(q, e, w),
        at(Where)
    {}

String TrcQInsertAt::Description () const
    {
        strstream ss;
        ss  << txtInserts << ' ' << TxtItselfIfCurrent (obj1)
            << txtInto << ' ' << Quote (queue.Name())
            << at << ' ' << Quote (obj2.Name()) << ends;
        return ss;
    }

// ---------------------------------------------------------------------------

TrcQRemove::TrcQRemove (const QueueBased& q, const Entity& e)
    :   TrcQMessage(q, e, ModelComponent::NullEntity())
    {}

String TrcQRemove::Description () const
    {
        strstream ss;
        ss  << txtRemoves << " " << TxtItselfIfCurrent (obj1)
             << ' ' << txtFrom << ' ' << Quote (queue.Name()) << ends;
        return ss;
    }

// ---------------------------------------------------------------------------

TrcQFind::TrcQFind (const QueueBased& q, const Entity& e)
    :   TrcQMessage(q, e, ModelComponent::NullEntity())
    {}

String TrcQFind::Description () const
    {
        strstream ss;
```

```
        ss  << txtFinds << " " << TxtItselfIfCurrent (obj1)
            << txtIn << ' ' << Quote (queue.Name()) << ends;
        return ss;
    }

// ----------------------------------------------------------------------------
```

msgresbi.h

```
// ----------------------------------------------------------------------------
//
// Datei
//          msgresbi.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ----------------------------------------------------------------------------
//
// Beschreibung
//
//          Nachrichten, die von Res und Bin gesendet werden
//
// ----------------------------------------------------------------------------

#ifndef MSGRESBIN_H
#define MSGRESBIN_H

// ----------------------------------------------------------------------------

#include "msgtypes.h"   // Basisklasse

// ----------------------------------------------------------------------------
// ----------------------------------------------------------------------------

class QueueBased;

// ----------------------------------------------------------------------------

class TrcResBin : public TraceMessage
{
    protected:
                    TrcResBin (const QueueBased& qb, unsigned long n = 0);

        const   QueueBased& resBin;
        unsigned long       n;
};

// ----------------------------------------------------------------------------

class TrcResBinAwait : public TrcResBin
{
    public:
                    TrcResBinAwait (const QueueBased& bin, unsigned long n);

        virtual String  Description () const;
};

// ----------------------------------------------------------------------------
// ----------------------------------------------------------------------------

class Bin;

// ----------------------------------------------------------------------------

class TrcBinGive : public TrcResBin
{
    public:
                    TrcBinGive (const Bin& bin, unsigned long n);
```

```
        virtual String  Description () const;
};

// --------------------------------------------------------------------------

class TrcBinTake : public TrcResBin
{
    public:
                    TrcBinTake (const Bin& bin, unsigned long n);

        virtual String  Description () const;
};

// --------------------------------------------------------------------------
// --------------------------------------------------------------------------

class Res;

// --------------------------------------------------------------------------

class TrcResSeize : public TrcResBin
{
    public:
                    TrcResSeize (const Res& res, unsigned long n);

        virtual String  Description () const;
};

// --------------------------------------------------------------------------

class TrcResRelease : public TrcResBin
{
    public:
                    TrcResRelease (const Res& res, unsigned long n);

        virtual String  Description () const;
};

// --------------------------------------------------------------------------

#endif  // MSGRESBIN_H
```

msgresbi.cc

```
// --------------------------------------------------------------------------
//
// Datei
//         msgresbi.cc
//
// Diplomarbeit
//
//         DESMO-C
//         Implementierung eines Simulators fuer
//         zeitdiskrete Simulation in C++
//
// Autor
//         Thomas Schniewind
//
// Datum
//         8.3.1998
//
// --------------------------------------------------------------------------

#include "msgresbi.h"

#include "bin.h"
#include "res.h"
#include "text.h"

// --------------------------------------------------------------------------
// fuer Res und Bin
// --------------------------------------------------------------------------

TrcResBin::TrcResBin (const QueueBased& qb, unsigned long N)
    :   TraceMessage    (),
        resBin          (qb),
        n               (N)
```

```
    {}

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

TrcResBinAwait::TrcResBinAwait (const QueueBased& b, unsigned long n)
    :   TrcResBin   (b, n)
    {}

// ---------------------------------------------------------------------------

String TrcResBinAwait::Description () const
    {
        strstream ss;
        ss  << txtAwaits << ' ' << n << ' ' << txtOf << ' '
            << resBin.QuotedName() << ends;
        return ss;
    }

// ---------------------------------------------------------------------------
// Bin
// ---------------------------------------------------------------------------

TrcBinGive::TrcBinGive (const Bin& b, unsigned long n)
    :   TrcResBin   (b, n)
    {}

// ---------------------------------------------------------------------------

String TrcBinGive::Description () const
    {
        strstream ss;
        ss  << txtGives << ' ';
        if (n <= 0)
            ss  << txtNone << " [" << n << "] ";
        else
            ss  << n << ' ';
        ss  << txtTo << ' ' << resBin.QuotedName() << ends;
        return ss;
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

TrcBinTake::TrcBinTake (const Bin& b, unsigned long n)
    :   TrcResBin   (b, n)
    {}

// ---------------------------------------------------------------------------

String TrcBinTake::Description () const
    {
        strstream ss;
        ss  << txtTakes << ' ';
        if (n <= 0)
            ss  << txtNone << " [" << n << "] ";
        else
            ss  << n << ' ';
        ss  << txtFrom << ' ' << resBin.QuotedName() << ends;
        return ss;
    }

// ---------------------------------------------------------------------------
// Res
// ---------------------------------------------------------------------------

TrcResSeize::TrcResSeize (const Res& r, unsigned long n)
    :   TrcResBin   (r, n)
    {}

// ---------------------------------------------------------------------------

String TrcResSeize::Description () const
    {
        strstream ss;
        ss  << txtSeizes << ' ';
        if (n <= 0)
            ss  << txtNone << " [" << n << "] ";
        else
            ss  << n << ' ';
        ss  << txtOf << ' ' << resBin.QuotedName() << ends;
        return ss;
    }
```

```
// --------------------------------------------------------------------------
// --------------------------------------------------------------------------

TrcResRelease::TrcResRelease (const Res& r, unsigned long n)
    :   TrcResBin   (r, n)
    {}

// --------------------------------------------------------------------------

String TrcResRelease::Description () const
    {
        strstream ss;
        ss  << txtReleases << ' ';
        if (n <= 0)
            ss  << txtNone << " [" << n << "] ";
        else
            ss  << n << ' ';
        ss  << txtTo << ' ' << resBin.QuotedName() << ends;
        return ss;
    }

// --------------------------------------------------------------------------
```

msgsched.h

```
// --------------------------------------------------------------------------
//
// Datei
//          msgsched.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// --------------------------------------------------------------------------
//
// Beschreibung
//
//          alle Nachrichten, die von Schedulables und deren Unterklassen
//          gesendet werden
//
// --------------------------------------------------------------------------

#ifndef MSGSCEDULABLE_H
#define MSGSCEDULABLE_H

// --------------------------------------------------------------------------

#include "msgtypes.h"

#include "entity.h"
#include "process.h"

// --------------------------------------------------------------------------
// Trace-Meldungen von Events und Entities
// --------------------------------------------------------------------------

class TrcSchedule : public TraceMessage
{
    public:
                TrcSchedule (   const SimTime&  dt,
                                const Event&    ev,
                                const Entity&   en = ModelComponent::
                                                        NullEntity());

        virtual String  Description () const;

    protected:
        const   SimTime     dt;
        const   Event&      event;
```

```
        const   Entity&     entity;
};

// ---------------------------------------------------------------------------

class TrcScheduleAfter : public TraceMessage
{
    public:
                TrcScheduleAfter (  const Schedulable&  before,
                                    const Event&        ev,
                                    const Entity&       en = ModelComponent::
                                                                NullEntity());

        virtual String  Description () const;

    protected:
        const   Event&          event;
        const   Schedulable&    after;
        const   Entity&         entity;
};

// ---------------------------------------------------------------------------

class TrcScheduleBefore : public TraceMessage
{
    public:
                TrcScheduleBefore ( const Schedulable&  before,
                                    const Event&        ev,
                                    const Entity&       en = ModelComponent::
                                                                NullEntity());

        virtual String  Description () const;

    protected:
        const   Event&          event;
        const   Schedulable&    before;
        const   Entity&         entity;
};

// ---------------------------------------------------------------------------

class TrcReSchedule : public TraceMessage
{
    public:
                TrcReSchedule ( const SimTime&      dt,
                                const Schedulable&  s);

        virtual String  Description () const;

    protected:
        const   SimTime         dt;
        const   Schedulable&    schedulable;
};

// ---------------------------------------------------------------------------

class TrcCancel : public TraceMessage
{
    public:
                TrcCancel (const Schedulable& s);

        virtual String  Description () const;
    protected:
        const   Schedulable&    schedulable;
};

// ---------------------------------------------------------------------------

class TrcDelete : public TraceMessage
{
    public:
                TrcDelete ( const Entity&   en);

        virtual String  Description () const;

    protected:
        const   Entity&     entity;
};

// ---------------------------------------------------------------------------
// Trace-Meldungen von Prozessen
// ---------------------------------------------------------------------------

class TrcTerminate : public TraceMessage
```

```
{
    public:
        virtual String  Description () const;
};

// ---------------------------------------------------------------------------

class TrcActivate : public TraceMessage
{
    public:
                TrcActivate (   const SimTime&  dt,
                                const Process&  p);

        virtual String  Description () const;

    protected:
        const   SimTime     dt;
        const   Process&    process;
};

// ---------------------------------------------------------------------------

class TrcReActivate : public TrcActivate
{
    public:
                TrcReActivate ( const SimTime&  dt,
                                const Process&  p);

        virtual String  Description () const;

};

// ---------------------------------------------------------------------------

class TrcActivateAfter : public TraceMessage
{
    public:
                TrcActivateAfter (  const Schedulable&  where,
                                    const Process&      p);

        virtual String  Description () const;

    protected:
        const   Schedulable&    where;
        const   Process&        process;
};

// ---------------------------------------------------------------------------

class TrcActivateBefore : public TrcActivateAfter
{
    public:
                TrcActivateBefore ( const Schedulable&  where,
                                    const Process&      p);

        virtual String  Description () const;
};

// ---------------------------------------------------------------------------

class TrcHold : public TraceMessage
{
    public:
                TrcHold (   const SimTime&  t);

        virtual String  Description () const;

    protected:
        const SimTime   time;
};

// ---------------------------------------------------------------------------

class TrcPassivate : public TraceMessage
{
    public:
        virtual String  Description () const;
};

// ---------------------------------------------------------------------------

class TrcInterrupt : public TraceMessage
{
    public:
```

```
                TrcInterrupt (  const Process&  p, const InterruptCode& ic);

        virtual String  Description () const;

    protected:
        const Process&          process;
        const InterruptCode&    ic;
};

// --------------------------------------------------------------------------

class TrcInterruptSlave : public TraceMessage
{
    public:
                TrcInterruptSlave ( const Process&  slave);

        virtual String  Description () const;

    protected:
        const Process&  slave;
};

// --------------------------------------------------------------------------

class ProcessQueue;
class ProcessCooperation;

class TrcCooperate : public TraceMessage
{
    public:
                TrcCooperate (const Process&            slave,
                              const ProcessQueue&       wq,
                              const ProcessCooperation& coop);

        virtual String  Description () const;

    protected:
        const Process&              slave;
        const ProcessQueue&         waitQueue;
        const ProcessCooperation&   coop;
};

// --------------------------------------------------------------------------

#endif  // MSGSCEDULABLE_H
```

msgsched.cc

```
// --------------------------------------------------------------------------
//
// Datei
//          msgsched.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// --------------------------------------------------------------------------

#include "msgsched.h"

#include "event.h"
#include "process.h"
#include "coop.h"
#include "interrup.h"
#include "pqueue.h"
#include "simtime.h"
#include "str.h"
#include "strstr.h"     // fuer strstream
#include "text.h"
```

```
// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

TrcSchedule::TrcSchedule (const SimTime& DT,    const Event&  ev,
                                                const Entity& en)
    :   dt      (DT),
        event   (ev),
        entity  (en)
    {}

// ---------------------------------------------------------------------------

String TrcSchedule::Description () const
    {
        strstream ss;
        ss << txtSchedules << ' ' << event.QuotedName() << ' ';
        if (!entity.IsNullEntity())
            ss << txtOf << ' ' << TxtItselfIfCurrent (entity) << ' ';
        ss << TxtDtToAt (dt) << ends;
        return ss;
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

TrcScheduleAfter::TrcScheduleAfter (const Schedulable& After,
                                    const Event& ev,
                                    const Entity& en)
    :   event(ev),
        after(After),
        entity(en)
    {}

// ---------------------------------------------------------------------------

String TrcScheduleAfter::Description () const
    {
        SimTime at = after.ScheduledAt();
        strstream ss;
        ss << txtSchedules << ' ' << event.QuotedName() << ' ';
        if (!entity.IsNullEntity())
            ss << txtOf << ' ' << TxtItselfIfCurrent (entity) << ' ';
        ss  << txtAfter << ' ' << after.QuotedName() << ' '
            << TxtTimeToAt (after.ScheduledAt()) << ends;
        return ss;
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

TrcScheduleBefore::TrcScheduleBefore (const Schedulable& Before,
                                      const Event& ev,
                                      const Entity& en)
    :   event(ev),
        before(Before),
        entity(en)
    {}

// ---------------------------------------------------------------------------

String TrcScheduleBefore::Description () const
    {
        strstream ss;
        ss << txtSchedules << ' ' << Quote (event.Name()) << ' ';
        if (!entity.IsNullEntity())
            ss << txtOf << ' ' << TxtItselfIfCurrent (entity) << ' ';
        ss  << txtBefore << ' ' << before.QuotedName() << ' '
            << TxtTimeToAt (before.ScheduledAt()) << ends;
        return ss;
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

TrcCancel::TrcCancel (const Schedulable& s)
    :   schedulable (s)
    {}

// ---------------------------------------------------------------------------

String TrcCancel::Description () const
    {
        strstream ss;
```

```
        ss  << txtCancels << ' ' << schedulable.QuotedName() << ends;
        return ss;
    }

// --------------------------------------------------------------------------
// --------------------------------------------------------------------------

TrcReSchedule::TrcReSchedule (const SimTime& DT,    const Schedulable&  s)
    :   dt          (DT),
        schedulable (s)
    {}

// --------------------------------------------------------------------------

String TrcReSchedule::Description () const
    {
        strstream ss;
        ss << txtReSchedules << ' ' << schedulable.QuotedName() << ' ';
        ss << TxtDtToAt (dt) << ends;
        return ss;
    }

// --------------------------------------------------------------------------
// --------------------------------------------------------------------------

TrcDelete::TrcDelete (const Entity& en)
    :   entity  (en)
    {}

// --------------------------------------------------------------------------

String TrcDelete::Description () const
    {
        strstream ss;
        ss << txtDeletes << ' ' << TxtItselfIfCurrent (entity) << ends;
        return ss;
    }

// --------------------------------------------------------------------------
// Process
// --------------------------------------------------------------------------

String TrcTerminate::Description () const
    {
        return txtTerminates;
    }

// --------------------------------------------------------------------------
// --------------------------------------------------------------------------

TrcActivate::TrcActivate (const SimTime& DT, const Process& p)
    :   dt      (DT),
        process (p)
    {}

// --------------------------------------------------------------------------

String TrcActivate::Description () const
    {
        strstream ss;
        ss  << txtActivates << ' ' << TxtItselfIfCurrent (process) << ' '
            << TxtDtToAt (dt) << ends;
        return ss;
    }

// --------------------------------------------------------------------------
// --------------------------------------------------------------------------

TrcReActivate::TrcReActivate (const SimTime& dt, const Process& p)
    :   TrcActivate     (dt, p)
    {}

// --------------------------------------------------------------------------

String TrcReActivate::Description () const
    {
        strstream ss;
        ss  << txtReActivates << ' ' << TxtItselfIfCurrent (process) << ' '
            << TxtDtToAt (dt) << ends;
        return ss;
    }

// --------------------------------------------------------------------------
// --------------------------------------------------------------------------
```

```
TrcActivateAfter::TrcActivateAfter (const Schedulable& s, const Process& p)
    :   where   (s),
        process (p)
    {}

// ---------------------------------------------------------------------------

String TrcActivateAfter::Description () const
    {
        strstream ss;
        ss  << txtActivates << ' ' << TxtItselfIfCurrent (process) << ' '
            << txtAfter << ' ' << TxtItselfIfCurrent (where) << ends;
        return ss;
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

TrcActivateBefore::TrcActivateBefore (const Schedulable& s, const Process& p)
    :   TrcActivateAfter    (s, p)
    {}

// ---------------------------------------------------------------------------

String TrcActivateBefore::Description () const
    {
        strstream ss;
        ss  << txtActivates << ' ' << TxtItselfIfCurrent (process) << ' '
            << txtBefore << ' ' << TxtItselfIfCurrent (where) << ends;
        return ss;
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

TrcHold::TrcHold (const SimTime& t)
    :   time    (t)
    {}

// ---------------------------------------------------------------------------

String TrcHold::Description () const
    {
        strstream ss;
        ss  << txtHoldsFor << ' ' << time << ", " << txtUntil
            << ' ' << (Time() + time) << ends;
        return ss;
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

String TrcPassivate::Description () const
    {
        return txtPassivates;
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

TrcInterrupt::TrcInterrupt (const Process& p, const InterruptCode& IC)
    :   process (p),
        ic      (IC)
    {}

// ---------------------------------------------------------------------------

String TrcInterrupt::Description () const
    {
        strstream ss;
        ss  << txtInterrupts << ' ' << TxtItselfIfCurrent (process)
            << ", " << txtWithReason << ' ';
        if (ic.Name().Length() > 0)
            ss  << ic.QuotedName() << ' ';
        ss << '[' << ic.GetCode() << ']' << ends;
        return ss;
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

TrcInterruptSlave::TrcInterruptSlave (const Process& s)
    :   slave   (s)
```

```
    {}

// ---------------------------------------------------------------------------

String TrcInterruptSlave::Description () const
    {
        strstream ss;
        ss  << txtInterrupts << ' ' << TxtItselfIfCurrent (slave)
            << ", " << txtInterruptWho << ends;
        return ss;
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

TrcCooperate::TrcCooperate (const Process&              s,
                            const ProcessQueue&         wq,
                            const ProcessCooperation&   c)
    :   slave       (s),
        waitQueue   (wq),
        coop        (c)
    {}

// ---------------------------------------------------------------------------

String TrcCooperate::Description () const
    {
        strstream ss;
        ss  << txtCoOpts << ' ' << TxtItselfIfCurrent (slave) << ' '
            << txtFrom << ' ' << waitQueue.QuotedName();

        if (coop.Name().Length() > 0 && coop.ShowInTrace())
            ss  << ' ' << txtCoOptsFor << ' ' << coop.QuotedName();

        ss << ends;
        return ss;
    }

// ---------------------------------------------------------------------------
```

msgtypes.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          msgtypes.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------
//
// Beschreibung
//
//          alle Nachrichtentypen, die vom System unterschieden werden
//
// ---------------------------------------------------------------------------

#ifndef MESSAGETYPES_H
#define MESSAGETYPES_H

// ---------------------------------------------------------------------------

#include "message.h"    // Basisklasse
#include "msgcomp.h"    // Hints & Consequences
#include "str.h"

// ---------------------------------------------------------------------------
// Meldung, die einen Hinweis auf die Stelle im Programm enthaelt
```

```
// bleibt abstrakt

class LocatableMessage : public Message
{
    public:
                LocatableMessage (MessageType, const String& where,
                                  const MsgConsequence* consequence = 0,
                                  const MsgHint*        hint        = 0);
                LocatableMessage (MessageType, CodeType, const String& where,
                                  const MsgConsequence* consequence = 0,
                                  const MsgHint*        hint        = 0);

        virtual ~LocatableMessage();
                    //  loescht consequence und hint

        virtual String          Location () const;
        virtual String          Consequences () const;
        virtual String          Hint () const;
    protected:
        const   String          where;
        const   MsgConsequence* consequence;
        const   MsgHint*        hint;
};

// ---------------------------------------------------------------------------
// Fehlermeldung, die zum sofortigen Programmabbruch fuehrt
// bleibt abstrakt

class FatalErrorMessage : public LocatableMessage
{
    public:
        FatalErrorMessage ( const String&           where,
                            const MsgConsequence*   consequence = 0,
                            const MsgHint*          hint        = 0);
};

// ---------------------------------------------------------------------------
// Fehlermeldung, die zum kontrollierten Programmabbruch fuehrt
// bleibt abstrakt

class ErrorMessage : public LocatableMessage
{
    public:
                    ErrorMessage (  const String& where,
                                    const MsgConsequence*   consequence = 0,
                                    const MsgHint*          hint        = 0);
};

// ---------------------------------------------------------------------------
// Fehlermeldung, die nicht zum Programmabbruch fuehrt, bleibt abstrakt

class WarningMessage : public LocatableMessage
{
    public:
                    WarningMessage (const String&           where,
                                    const MsgConsequence*   consequence = 0,
                                    const MsgHint*          hint        = 0);
};

// ---------------------------------------------------------------------------
// globale (experimentuebergreifende) Fehlermeldung, bleibt abstrakt

class GlobalErrorMessage : public LocatableMessage
{
    public:
                GlobalErrorMessage (const String& where,
                                    CodeType ct = normalError,
                                    const MsgConsequence* consequence = 0,
                                    const MsgHint*        hint      = 0);
};

// ---------------------------------------------------------------------------
// Meldung an den Ergebnis-Report, bleibt abstrakt

class ReportMessage : public Message
{
    public:
                ReportMessage (const String& what, CodeType ct = normal);
        virtual String      Description () const;
    private:
        const   String  what;
};

// ---------------------------------------------------------------------------
```

```
// Nachrichten an den Trace-Manager, bleibt abstrakt

class TraceMessage : public Message
{
    public:
                TraceMessage (CodeType ct = normal);
};

// ------------------------------------------------------------------------

class CustomTraceMessage : public TraceMessage
{
    public:
                CustomTraceMessage  (const String& description = "");
                CustomTraceMessage  (       CodeType ct,
                                    const   String&  description = "");

        virtual String  Description () const;

    protected:
        const   String      description;
};

// ------------------------------------------------------------------------
// Nachrichten an den Debug-Manager

class DebugMessage : public Message
{
    public:
                DebugMessage (const String& descr, CodeType ct = normal);

        virtual String      Description () const;
    private:
        String  description;
};

// ------------------------------------------------------------------------

#endif  // MESSAGETYPES_H
```

msgtypes.cc

```
// ------------------------------------------------------------------------
//
// Datei
//          msgtypes.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ------------------------------------------------------------------------

#include "msgtypes.h"

#include "str.h"
#include "strstr.h"
#include "text.h"

// ------------------------------------------------------------------------
// ------------------------------------------------------------------------
// Klasse LocatableMessage:

LocatableMessage::LocatableMessage (    MessageType mt,
                                        const String& w,
                                        const MsgConsequence*   c,
                                        const MsgHint*          h)
    :   Message     (mt),
        where       (w),
        consequence (c),
```

```
        hint        (h)
    {}

// ---------------------------------------------------------------------------

LocatableMessage::LocatableMessage (    MessageType mt,
                                        CodeType ct,
                                        const String& w,
                                        const MsgConsequence*   c,
                                        const MsgHint*          h)
    :   Message     (mt, ct),
        where       (w),
        consequence (c),
        hint        (h)
    {}

// ---------------------------------------------------------------------------

LocatableMessage::~LocatableMessage ()
    {
        if (consequence)    delete consequence;
        if (hint)           delete hint;
    }

// ---------------------------------------------------------------------------

String LocatableMessage::Location () const
    {   return where; }

// ---------------------------------------------------------------------------

String LocatableMessage::Consequences () const
    {
        if (consequence)    return consequence->Consequences();
        else                return "";
    }

// ---------------------------------------------------------------------------

String LocatableMessage::Hint () const
    {
        if (hint)   return hint->Hint();
        else        return "";
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

FatalErrorMessage::FatalErrorMessage (  const String&           w,
                                        const MsgConsequence*   c,
                                        const MsgHint*          h)
    :   LocatableMessage (error, fatalError, w, c, h)
    {}

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

ErrorMessage::ErrorMessage (    const String&           w,
                                const MsgConsequence*   c,
                                const MsgHint*          h)
    :   LocatableMessage (error, normalError, w, c, h)
    {}

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

WarningMessage::WarningMessage (const String&           w,
                                const MsgConsequence*   c,
                                const MsgHint*          h)
    :   LocatableMessage (error, warning, w, c, h)
    {}

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

GlobalErrorMessage::GlobalErrorMessage (const String&           w,
                                        CodeType                ct,
                                        const MsgConsequence*   c,
                                        const MsgHint*          h)
    :   LocatableMessage (globalError, ct, w, c, h)
    {}

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------
```

```
ReportMessage::ReportMessage (const String& w, CodeType ct)
    :   Message (report, ct),
        what(w)
    {}

// ---------------------------------------------------------------------------

String ReportMessage::Description () const
    {   return what; }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

TraceMessage::TraceMessage (CodeType ct)
    : Message (trace, ct)
    {}

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

CustomTraceMessage::CustomTraceMessage (const String& descr)
    :   description (descr)
    {}

// ---------------------------------------------------------------------------

CustomTraceMessage::CustomTraceMessage (CodeType ct, const String& descr)
    :   TraceMessage    (ct),
        description     (descr)
    {}

// ---------------------------------------------------------------------------

String CustomTraceMessage::Description () const
    {
        return description;
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

DebugMessage::DebugMessage (const String& s, CodeType ct)
    :   Message     (debug, ct),
        description (s)
    {}

// ---------------------------------------------------------------------------

String DebugMessage::Description () const
    {
        return description;
    }

// ---------------------------------------------------------------------------
```

msgwaitq.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          msgwaitq.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------
//
// Beschreibung
```

```
//
//          Nachrichten, die von WaitQueues gesendet werden
//
// ---------------------------------------------------------------------------

#ifndef MSGWAITQ_H
#define MSGWAITQ_H

// ---------------------------------------------------------------------------

#include "msgtypes.h"   // Basisklasse

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

class QueueBased;
class ProcessCooperation;
class Condition;

// ---------------------------------------------------------------------------

class TrcWaitQ : public TraceMessage
{
   protected:
                    TrcWaitQ (const QueueBased& wq);

        const   QueueBased& waitQueue;
};

// ---------------------------------------------------------------------------

class TrcWaitQWait : public TrcWaitQ
{
   public:
                    TrcWaitQWait (const QueueBased& wq);

        virtual String  Description () const;
};

// ---------------------------------------------------------------------------

class TrcWaitQWaitFor : public TrcWaitQ
{
   public:
                    TrcWaitQWaitFor (const QueueBased& wq,
                                     const Condition&  c);

        virtual String  Description () const;
   protected:
        const   Condition&  cond;
};

// ---------------------------------------------------------------------------

class TrcWaitQFind : public TrcWaitQ
{
   public:
                    TrcWaitQFind (const Process&            slave,
                                  const QueueBased&         wq,
                                  const ProcessCooperation& coop,
                                  const Condition&          cond);

        virtual String  Description () const;
   protected:
        const   Process&            slave;
        const   ProcessCooperation& coop;
        const   Condition&          cond;
};

// ---------------------------------------------------------------------------

#endif  // MSGWAITQ_H
```

msgwaitq.cc

```
// ---------------------------------------------------------------------------
//
// Datei
```

```
//          msgwaitq.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// -----------------------------------------------------------------------------

#include "msgwaitq.h"

#include "qbased.h"
#include "process.h"
#include "coop.h"
#include "conditio.h"
#include "text.h"

// -----------------------------------------------------------------------------
// -----------------------------------------------------------------------------

TrcWaitQ::TrcWaitQ (const QueueBased& wq)
    :   TraceMessage    (),
        waitQueue       (wq)
    {}

// -----------------------------------------------------------------------------
// -----------------------------------------------------------------------------

TrcWaitQWait::TrcWaitQWait (const QueueBased& wq)
    :   TrcWaitQ    (wq)
    {}

// -----------------------------------------------------------------------------

String TrcWaitQWait::Description () const
    {
        strstream ss;
        ss  << txtWaitsIn << ' ' << waitQueue.QuotedName() << ends;
        return ss;
    }

// -----------------------------------------------------------------------------
// -----------------------------------------------------------------------------

TrcWaitQWaitFor::TrcWaitQWaitFor (const QueueBased& wq,
                                  const Condition&  c)
    :   TrcWaitQ    (wq),
        cond        (c)
    {}

// -----------------------------------------------------------------------------

String TrcWaitQWaitFor::Description () const
    {
        strstream ss;
        ss  << txtWaitsIn << ' ' << waitQueue.QuotedName();

        if (cond.Name().Length() > 0 && cond.ShowInTrace())
            ss << ' ' << txtWaitsFor << ' ' << cond.QuotedName();

        ss  << ends;
        return ss;
    }

// -----------------------------------------------------------------------------
// -----------------------------------------------------------------------------

TrcWaitQFind::TrcWaitQFind (const Process&              s,
                            const QueueBased&           wq,
                            const ProcessCooperation&   cooperation,
                            const Condition&            condition)
    :   TrcWaitQ    (wq),
        slave       (s),
        coop        (cooperation),
        cond        (condition)
    {}
```

```
// --------------------------------------------------------------------------

String TrcWaitQFind::Description () const
    {
        strstream ss;
        ss  << txtFinds << ' ';

        if (cond.Name().Length() > 0 && cond.ShowInTrace())
            ss << cond.QuotedName() << ' ';

        ss  << slave.QuotedName() << ' '
            << txtIn << ' ' << waitQueue.QuotedName();

        if (coop.Name().Length() > 0 && coop.ShowInTrace())
            ss  << ' ' << txtCoOptsFor << ' ' << coop.QuotedName();

        ss << ends;
        return ss;
    }

// --------------------------------------------------------------------------
```

namecat.h

```
// --------------------------------------------------------------------------
//
// Datei
//          namecat.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// --------------------------------------------------------------------------
//
// Weiterentwicklung von:
//
// Diplomarbeit
//
//          Entwurf und Realisierung eines objektorientierten
//          Simulationspakets in C++
//
// Author
//          Heiko Weber
//
// Beschreibung
//
//          Die Klasse EntityCatalog dient als Nachschlage-Katalog
//          fuer die verwendeten Entity-Namen.
//
// --------------------------------------------------------------------------

#ifndef  NAMECATALOG_H
#define  NAMECATALOG_H

// --------------------------------------------------------------------------

#include "avl.h"

#include "str.h"

// --------------------------------------------------------------------------

class NameCatalog : private Avl
{
    public:
                NameCatalog (unsigned nameWith,
                             unsigned noWidth,
                             char     fill = ' ');
        virtual ~NameCatalog();
```

```
        String      AddNumberTo     (const String& name);
                        // haengt an name eine Seriennr.

    private:
        unsigned    GetNumberOf     (const String& name);
                        // liefert die naechste Serienr.

        unsigned    nameWidth;  // Stringbreite fuer den Namen
        unsigned    noWidth;    // Stringbreite fuer die Nummer
        unsigned    modulo;     // 0 < Nummer < modulo
        char        fill;       // Fuellzeichen fuer ungenutze Stellen
};

// ----------------------------------------------------------------------------

#endif  // NAMECATALOG_H
```

namecat.cc

```
// ----------------------------------------------------------------------------
//
// Datei
//          namecat.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ----------------------------------------------------------------------------
//
// Weiterentwicklung von:
//
// ecatalog.cc
//
// Diplomarbeit
//
//          Entwurf und Realisierung eines objektorientierten
//          Simulationspakets in C++
//
// Author
//          Heiko Weber
//
// Beschreibung
//
//          Die Klasse EntityCatalog dient als Nachschlage-Katalog
//          fuer die verwendeten Entity-Namen.
//
// ----------------------------------------------------------------------------

#include "namecat.h"

#include "math.h"       // fuer log10()
#include "nobject.h"
#include "str.h"

// ----------------------------------------------------------------------------

class NameCatalogItem : public NamedObject
{
    public:
                NameCatalogItem (const String& name)
                    : NamedObject(name), count(1) {}
        virtual ~NameCatalogItem() {}

        int     GetCount() { return count; }
        int     SetCount (int n)
                    { int o = count; count = n; return o; }

        static AvlCmpResult Compare (const void*, const void*);
```

```
    private:
        int     count;
};

// ---------------------------------------------------------------------------

AvlCmpResult NameCatalogItem::Compare (const void *v1, const void *v2)
    {
        const NameCatalogItem*  i1  = (const NameCatalogItem*) v1;
        const NameCatalogItem*  i2  = (const NameCatalogItem*) v2;
        const int               cmp = i1->Name().Compare (i2->Name());

        return (!cmp) ? EQUAL : (cmp < 0) ? LESS : GREATER;
    }

// ---------------------------------------------------------------------------

NameCatalog::NameCatalog (unsigned naw, unsigned now, char fillC)
    :   Avl         (NameCatalogItem::Compare),
        nameWidth   (naw - now),
        noWidth     (now),
        modulo      (unsigned (pow(10, noWidth))),
        fill        (fillC)
    {}

// ---------------------------------------------------------------------------

NameCatalog::~NameCatalog ()
    {
        void *tmp;

        while ((tmp = First()) != 0)
        {
            Remove (tmp);
            delete (NameCatalogItem*) tmp;
        }
    }

// ---------------------------------------------------------------------------

unsigned NameCatalog::GetNumberOf (const String& name)
    {
        NameCatalogItem*    tmp1 = new NameCatalogItem (name);
        NameCatalogItem*    tmp2;
        unsigned            cnt  = 1;

        if ((tmp2 = (NameCatalogItem*) Search (tmp1)) != 0)
        {
            tmp2->SetCount (cnt = (tmp2->GetCount() + 1) % modulo);
            if (cnt == 0)
                tmp2->SetCount (cnt = 1);
            delete tmp1;
        } else
            Insert(tmp1);

        return cnt;
    }

// ---------------------------------------------------------------------------

String NameCatalog::AddNumberTo (const String& name)
    {
        // die nummer wird ueber 'name' ermittelt, nicht ueber
        // die gekuerzte Version!!!
        String      no  = GetNumberOf (name);
        unsigned    noL = no.Length();
        String      n   = String (name, nameWidth);

        // nol <= noWidth
        if (noL < noWidth)
            return n + String (fill, noWidth - noL) + no;
        else
            return n + no;
    }

// ---------------------------------------------------------------------------
```

nobject.h

```
// -------------------------------------------------------------------------
//
// Datei
//          nobject.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// -------------------------------------------------------------------------
//
// Beschreibung
//
//          Die Klasse NamedObject hat die Eigenschaft, dass jedes Onjekt
//          dieser Klasse einen eigenen Namen erhalten kann.
//          Dieser kann sowohl bei der Erzeugung (Konstruktor), als auch
//          spaeter vergeben oder geaendert werden (Rename()). Mit Hilfe
//          dieser Klasse wird eine Benennung der Simulationsobjekte
//          vorgenommen.
//
// -------------------------------------------------------------------------

#ifndef  NAMEDOBJECT_H
#define  NAMEDOBJECT_H

// -------------------------------------------------------------------------

#include "str.h"    // class String

// -------------------------------------------------------------------------

class NamedObject
{
    public:
        // Konstruktoren
                NamedObject (const String& s = "");
                NamedObject (const NamedObject&);
        // Destruktor
        virtual ~NamedObject ();

        // Selektoren
        const   String& Name () const;
                String  Name ();
                String  QuotedName () const;
        virtual String  ClassName() const;

                bool    Valid () const;
                        /*  true, wenn selfRef auf das Objekt selbst zeigt */

        // Manipulatoren
        virtual void    Rename (const String& s);
                            /*  virtuell, damit Unterklassen auf eine Namens-
                                aenderung reagieren koenne. Diese sollten
                                immer die geerbte Version von Rename()
                                aufrufen!
                            */

        NamedObject&    operator=(const NamedObject&);

    private:
        String          name;
        NamedObject*    selfRef;
};

// -------------------------------------------------------------------------

#endif //NAMEDOBJECT_H
```

nobject.cc

```
// --------------------------------------------------------------------------
//
// Datei
//          nobject.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// --------------------------------------------------------------------------

#include "nobject.h"

// --------------------------------------------------------------------------

static const char* className = "NamedObject";

// --------------------------------------------------------------------------

NamedObject::NamedObject (const String& s)
    :   name (s),
        selfRef (this)
    {}

// --------------------------------------------------------------------------
//

NamedObject::NamedObject (const NamedObject& n)
    :   name    (n.name),
        selfRef (this)
    {}

// --------------------------------------------------------------------------

NamedObject::~NamedObject ()
    {
        selfRef = 0;
    }

// --------------------------------------------------------------------------
// Selektoren

const String& NamedObject::Name () const
    {
        return name;
    }

// --------------------------------------------------------------------------

String NamedObject::Name ()
    {
        return String(name);
    }

// --------------------------------------------------------------------------

String NamedObject::QuotedName () const
    {
        return String('\'') + name + '\'';
    }

// --------------------------------------------------------------------------

String NamedObject::ClassName () const
    {
        return className;
    }

// --------------------------------------------------------------------------

bool NamedObject::Valid () const
    {
```

```
        return this == selfRef;
    }

// ------------------------------------------------------------------------
// Manipulatoren

void NamedObject::Rename (const String& s)
    {
        name = s;
    }

// ------------------------------------------------------------------------

NamedObject& NamedObject::operator= (const NamedObject& no)
    {
        Rename (no.name);
        return *this;
    }

// ------------------------------------------------------------------------
```

nullobj.h

```
// ------------------------------------------------------------------------
//
// Datei
//          nullobj.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ------------------------------------------------------------------------

#ifndef NULLOBJECTS_H
#define NULLOBJECTS_H

// ------------------------------------------------------------------------

#include "experime.h"   // Basisklasse
#include "model.h"      // Basisklasse
#include "event.h"      // Basisklasse
#include "process.h"    // Basisklasse

// ------------------------------------------------------------------------
// DefaultModel

class DefaultModel : public Model
{
        // nicht implementiert:
                        DefaultModel    (const DefaultModel&);
        DefaultModel&   operator=       (const DefaultModel&);

    public:
                DefaultModel ();
};

// ------------------------------------------------------------------------
// Klasse TheNullEvent:

class TheNullEvent : public Event
{
        // nicht implementiert:
                        TheNullEvent    (const TheNullEvent&);
        TheNullEvent&   operator=       (const TheNullEvent&);

    public:
                TheNullEvent (Model& owner);
        virtual ~TheNullEvent ();
```

```
        virtual bool    Valid() const;

    protected:
        virtual void    EventRoutine (Entity& entity);
};

// --------------------------------------------------------------------------
// TheNullProcess

class TheNullProcess : public Process
{
        // nicht implementiert:
                        TheNullProcess  (const TheNullProcess&);
        TheNullProcess& operator=       (const TheNullProcess&);

    public:
                TheNullProcess (Model& owner);
        virtual ~TheNullProcess ();

        virtual bool    Valid() const;

    protected:
        virtual void    LifeCycle ();
};

// --------------------------------------------------------------------------
// DefaultExperiment

class DefaultExperiment : public Experiment
{
        friend class ExperimentManager;

        // nicht implementiert:
                            DefaultExperiment   (const DefaultExperiment&);
        DefaultExperiment&  operator=           (const DefaultExperiment&);

    public:
                DefaultExperiment ();
    private:
        DefaultModel    defaultModel;
        TheNullEvent    nullEvent;
        TheNullProcess  nullProcess;
};

// --------------------------------------------------------------------------

#endif  // NULLOBJECTS_H
```

nullobj.cc

```
// --------------------------------------------------------------------------
//
// Datei
//          nullobj.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// --------------------------------------------------------------------------

#include "nullobj.h"

#include "experimm.h"

#include <assert.h>

// --------------------------------------------------------------------------
// DefaultModel
```

```
DefaultModel::DefaultModel ()
    :   Model   (0, "None")
    {}

// ---------------------------------------------------------------------------
// Klasse TheNullEvent:

TheNullEvent::TheNullEvent (Model& owner)
    :   Event   (owner, "")
    {
        // NameCatalog umgehen:
        NamedObject::Rename ("---");
    }

// ---------------------------------------------------------------------------

TheNullEvent::~TheNullEvent ()
    {
        const char* where = "TheNullEvent::~TheNullEvent";

        if (!ExperimentManager::Instance().InDeletion())
            Error ( "nullObjects must not be deleted!",
                    where, "experiment is aborted");
    }

// ---------------------------------------------------------------------------

bool TheNullEvent::Valid () const
    {
        return false;
    }

// ---------------------------------------------------------------------------

void TheNullEvent::EventRoutine (Entity& entity)
    {
        assert (false); /* Error! */
    }

// ---------------------------------------------------------------------------
// TheNullProcess

TheNullProcess::TheNullProcess (Model& owner)
    :   Process (owner, "")
    {
        // NameCatalog umgehen:
        NamedObject::Rename ("---");
    }

// ---------------------------------------------------------------------------

TheNullProcess::~TheNullProcess ()
    {
        const char* where = "TheNullProcess::~TheNullProcess";

        if (!ExperimentManager::Instance().InDeletion())
            FatalError ("nullObjects must not be deleted!",
                        where, "program is aborted");
    }

// ---------------------------------------------------------------------------

bool TheNullProcess::Valid () const
    {
        return false;
    }

// ---------------------------------------------------------------------------

void TheNullProcess::LifeCycle ()
    {
        // Fehlermeldung! NullProcess kann nicht aktiviert werden!
        assert (false);
    }

// ---------------------------------------------------------------------------
// DefaultExperiment

DefaultExperiment::DefaultExperiment ()
    :   Experiment("DESMO"),
        defaultModel(),
        nullEvent   (defaultModel),
        nullProcess (defaultModel)
    {}
```

```
// --------------------------------------------------------------------------
```

observab.h

```
// --------------------------------------------------------------------------
//
// Datei
//          observab.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// --------------------------------------------------------------------------

#ifndef OBSERVABLE_H
#define OBSERVABLE_H

// --------------------------------------------------------------------------

class Observer;
class ObserverList;

// --------------------------------------------------------------------------

class Observable
    /*  Observable setzt den Subjektteil des Beobachtermusters um.
        -> siehe Observer.
    */
{
        friend class Observer;  // fuer Register und Deregister

        Observable (const Observable& objToCopy);   // nicht implementiert
        Observable& operator= (const Observable&);  // nicht implementiert

    public:
                        Observable ();
        virtual        ~Observable ();

    protected:
                        void        NotifyObservers ();
                                        /*  Benachrichtigt alle angemeldeten
                                            Beobachter */
    private:
                        void        Register (Observer& o);
                        void        DeRegister (Observer& o);

        ObserverList*               observers;
                                        //  Liste der angemeldeten Beobachter
};

// --------------------------------------------------------------------------

#endif  // OBSERVABLE_H
```

observab.cc

```
// --------------------------------------------------------------------------
//
// Datei
//          observab.cc
//
// Diplomarbeit
```

```
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ----------------------------------------------------------------------------

#include "observab.h"

#include "observer.h"
#include "ring.h"

#include <assert.h>

// ----------------------------------------------------------------------------

class ObserverList : public Ring<Observer>
{};

// ----------------------------------------------------------------------------
// ----------------------------------------------------------------------------

Observable::Observable ()
    :   observers(new ObserverList)
    {}

// ----------------------------------------------------------------------------

Observable::~Observable ()
    {
        Observer*   obs;
        while ((obs = observers->Last()) != 0)
            obs->Observe (0);   // fuehrt zum entfernen
        delete observers;
    }

// ----------------------------------------------------------------------------

void Observable::Register (Observer& o)
    {
        if (!observers->Find (&o))
            observers->Append (&o);
    }

// ----------------------------------------------------------------------------

void Observable::DeRegister (Observer& o)
    {
        if (!observers->Remove (&o))
            assert(false);
    }

// ----------------------------------------------------------------------------

void Observable::NotifyObservers ()
    {
        // fuer den Fall, dass sich waehrenddessen ein Observer abmeldet,
        // wird die Observer-Liste kopiert, und auf dieser Kopie gearbeitet
        ObserverList ol = *observers;
        Observer* obs   = ol.First();

        for (int i = ol.Size(); i > 0 ; i--)
        {
            obs->NoteChange (this);
            obs = ol.Next();
        }
    }

// ----------------------------------------------------------------------------
```

observer.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          observer.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#ifndef OBSERVER_H
#define OBSERVER_H

// ---------------------------------------------------------------------------

class Observable;

// ---------------------------------------------------------------------------

class Observer
    /*  Observer setzt zusammen mit Observable das Beaobachtermuster um.
        Ein Observer registriert sich bei maximal einem Observable, ueber
        dessen Aenderungen er informiert werden moechte. */
    /*  Observable ruft fuer alle angemeldeten Observer bei einer Aenderung
        deren Methode NoteChange auf und uebergibt einen Zeiger auf sich
        selbst.
    */
{
        Observer&       operator= (const Observer&);    // nicht implementiert
    public:
                    Observer (Observable* obs = 0);
                    Observer (const Observer& objToCopy);
        virtual    ~Observer ();

        virtual void    NoteChange (Observable* obs) = 0;
                            /*  Implementiert die Reaktion auf die Aenderung
                                des Observables */

                void    Observe (Observable* obs = 0);
                            /*  obs wird von nun an beobachtet,
                                obs = 0 beendet die Beobachtung */

                bool    Observing () const;
                            /*  liefert true, falls ein Observable
                                beobachtet wird */
    protected:

    private:

        Observable*     observable;
};

// ---------------------------------------------------------------------------

#endif  // OBSERVER_H
```

observer.cc

```
// ---------------------------------------------------------------------------
//
// Datei
//          observer.cc
//
// Diplomarbeit
//
```

```
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// -----------------------------------------------------------------------------

#include "observer.h"

#include "observab.h"

// -----------------------------------------------------------------------------

Observer::Observer (Observable* obs)
    :   observable (obs)
    {
        if (observable)
            observable->Register (*this);
    }

// -----------------------------------------------------------------------------

Observer::Observer (const Observer& obs)
    :   observable (obs.observable)
    {
        if (observable)
            observable->Register (*this);
    }

// -----------------------------------------------------------------------------

Observer::~Observer ()
    {
        if (observable)
        {
            observable->DeRegister (*this);
            observable = 0;
        }
    }

// -----------------------------------------------------------------------------

void Observer::Observe (Observable* obs)
    {
        if (observable)
            observable->DeRegister (*this);
        if ((observable = obs) != 0)
            observable->Register (*this);
    }

// -----------------------------------------------------------------------------

bool Observer::Observing () const
    {
        return (observable != 0);
    }

// -----------------------------------------------------------------------------
```

outputm.h

```
// -----------------------------------------------------------------------------
//
// Datei
//          outputm.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
```

```
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// --------------------------------------------------------------------------

#ifndef OUTPUTMANAGER_H
#define OUTPUTMANAGER_H

// --------------------------------------------------------------------------

#include "messaged.h"   // Basisklasse

#include "stdoutp.h"

// --------------------------------------------------------------------------

class MessageManager;
class Experiment;

// --------------------------------------------------------------------------

class OutputManager : public MessageDistributor
{
    public:
                    OutputManager ( MessageManager&          mm,
                                    Experiment&              e,
                                    Message::MessageType    type);
        virtual    ~OutputManager ();

        virtual void    Note (const Message& msg);

                void    Add     (Output&);
                void    Remove  (Output&);

    protected:

    private:
                void    SwitchOn ();
                void    SwitchOff();
                Output& newStdOutput (  const   String&         name,
                                        Message::MessageType    t) const;

        MessageManager&         messageManager;
        Output&                 stdOutput;
        Message::MessageType    type;
};

// --------------------------------------------------------------------------

class GlobalErrorManager : public MessageDistributor
{
    public:
                    GlobalErrorManager (ostream& os);
        virtual    ~GlobalErrorManager ();

                void    Add     (Output&);
                void    Remove  (Output&);
    private:
        StdGlError      stdGlError;
};

// --------------------------------------------------------------------------

#endif  // OUTPUTMANAGER_H
```

outputm.cc

```
// --------------------------------------------------------------------------
//
// Datei
//          outputm.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
```

```
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#include "outputm.h"

#include "experime.h"
#include "experimm.h"
#include "messagem.h"
#include "message.h"

#include "stdoutp.h"

// ---------------------------------------------------------------------------

OutputManager::OutputManager (  MessageManager&         mm,
                                Experiment&             e,
                                Message::MessageType    t)
    :   messageManager  (mm),
        stdOutput       (newStdOutput (e.Name(), t)),
        type            (t)
    {
        Add (stdOutput);
        messageManager.Register (*this, type);
    }

// ---------------------------------------------------------------------------

OutputManager::~OutputManager ()
    {
        messageManager.DeRegister (*this);
        Remove (stdOutput);
        delete &stdOutput;
    }

// ---------------------------------------------------------------------------

void OutputManager::Note (const Message& msg)
    {
        int code = msg.Code();

        if (code == Message::switchOn && msg.Type() == type)
            SwitchOn();
        else
            if (code == Message::switchOff && msg.Type() == type)
                SwitchOff();

        MessageDistributor::Distribute (msg);
    }

// ---------------------------------------------------------------------------

void OutputManager::Add (Output& r)
    {
        Register (r, type);
        if (type == Message::trace)
            Register (r, Message::error);
        if (type == Message::debug)
        {
            Register (r, Message::error);
            Register (r, Message::trace);
        }
    }

// ---------------------------------------------------------------------------

void OutputManager::Remove (Output& r)
    {
        DeRegister (r);
    }

// ---------------------------------------------------------------------------

void OutputManager::SwitchOn()
    {
        if (type == Message::trace)
            messageManager.Register (*this, Message::error);
        if (type == Message::debug)
```

```
        {
            messageManager.Register (*this, Message::error);
            messageManager.Register (*this, Message::trace);
        }
    }

// ----------------------------------------------------------------------------

void OutputManager::SwitchOff ()
    {
        if (type == Message::trace)
            messageManager.DeRegister (*this, Message::error);
        if (type == Message::debug)
        {
            messageManager.DeRegister (*this, Message::error);
            messageManager.DeRegister (*this, Message::trace);
        }
    }

// ----------------------------------------------------------------------------

Output& OutputManager::newStdOutput (const  String&                 name,
                                            Message::MessageType    t) const
    {
        switch (t)
        {
            case Message::debug:
                return *new StdDebug (name, ".dbg");
            case Message::error:
                return *new StdError (name, ".err");
            case Message::globalError:
                return *new StdGlError (ExperimentManager::Instance().Err(),
                                        60);
            case Message::report:
                return *new StdReport (name, ".rpt");
            case Message::trace:
                return *new StdTrace (name, ".trc", 90);
            default:
                ;
        }
        return *new Output (ExperimentManager::Instance().Out());
    }

// ----------------------------------------------------------------------------
// ----------------------------------------------------------------------------

GlobalErrorManager::GlobalErrorManager (ostream& os)
    :   stdGlError      (os)
    {
        Add (stdGlError);
    }

// ----------------------------------------------------------------------------

GlobalErrorManager::~GlobalErrorManager ()
    {
        Remove (stdGlError);
    }

// ----------------------------------------------------------------------------

void GlobalErrorManager::Add (Output& r)
    {
        Register (r, Message::globalError);
    }

// ----------------------------------------------------------------------------

void GlobalErrorManager::Remove (Output& r)
    {
        DeRegister (r);
    }

// ----------------------------------------------------------------------------
```

pblocker.h

```
// ----------------------------------------------------------------------------
```

```
//
// Datei
//          pblocker.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#ifndef PROCESSBLOCKER_H
#define PROCESSBLOCKER_H

// ---------------------------------------------------------------------------

#include "boolean.h"
#include "process.h"

// ---------------------------------------------------------------------------

class ProcessBlocker
{       /*  ermoeglicht Implementierungsklassen den Zugriff auf das Blockade-
            Attribut von Prozessen, ohne 'Process' mit friend-Deklarationen
            zu ueberfrachten.
        */
    public:
        static  void    Block       (Process& p) {  p.blocked = true;
                                                    p.SkipTraceNote ();
                                                    p.Passivate();
                                                 }
        static  void    SetBlocked  (Process& p) {  p.blocked = true;  }
        static  void    UnBlock     (Process& p) {  p.blocked = false; }
};

// ---------------------------------------------------------------------------

#endif  // PROCESSBLOCKER_H
```

pimpl.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          pimpl.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#ifndef PROCESSIMPLEMENTATION_H
#define PROCESSIMPLEMENTATION_H

// ---------------------------------------------------------------------------

#include "coroutin.h"   // Basisklasse

// ---------------------------------------------------------------------------

class Process;
```

```
// ---------------------------------------------------------------------------

class ProcessImplementation : public Coroutine
{
                    ProcessImplementation (const ProcessImplementation&);
    public:
                    ProcessImplementation (Process& p);
        virtual    ~ProcessImplementation ();

    private:
        virtual void        Body ();

        Process&    process;
};

// ---------------------------------------------------------------------------

#endif  // PROCESSIMPLEMENTATION_H
```

pimpl.cc

```
// ---------------------------------------------------------------------------
//
// Datei
//          pimpl.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#include "pimpl.h"

#include "process.h"

// ---------------------------------------------------------------------------

ProcessImplementation::ProcessImplementation (Process& p)
    :   Coroutine(),
        process(p)
    {};

// ---------------------------------------------------------------------------

ProcessImplementation::~ProcessImplementation ()
    {};

// ---------------------------------------------------------------------------

void ProcessImplementation::Body ()
    {
        process.Start ();
    }

// ---------------------------------------------------------------------------
```

portable.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          portable.h
//
```

```
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ----------------------------------------------------------------------------

#ifndef PORTABLE_H
#define PORTABLE_H

// ----------------------------------------------------------------------------

// Anpassung der Bibliothek von Metrowerks CodeWarrior
#ifdef  __MWERKS__

#include <iostream.h>   // umm ggf. __MSL__ zu definieren

#ifdef  __MSL__

// ----------------------------------------------------------------------------

#else   // Plauger

#define tellp() rdbuf()->pubseekoff(0, ios::cur, ios::out).offset()
#define tellg() rdbuf()->pubseekoff(0, ios::cur, ios::in).offset()

#define seekp(pos) rdbuf()->pubseekpos(pos, ios::out)
#define seekg(pos) rdbuf()->pubseekpos(pos, ios::in)

#endif  // __MSL__
#endif  // __MWERKS__

// ----------------------------------------------------------------------------

#endif  // PORTABLE_H
```

pqueue.h

```
// ----------------------------------------------------------------------------
//
// Datei
//          pqueue.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ----------------------------------------------------------------------------

#ifndef PROCESSQUEUE_H
#define PROCESSQUEUE_H

// ----------------------------------------------------------------------------

#include "qbased.h" // Basisklasse

#include "str.h"

// ----------------------------------------------------------------------------

class Process;
class Condition;
class QueueImpl;
```

```
// --------------------------------------------------------------------------

class ProcessQueue : public QueueBased
    /*  ProcessQueue ist die Klasse der Warteschlangen, in denen Prozesse
        warten koenen. Entities, die keine Prozesse sind, koennen hier
        nicht eingereit werden.
    */
{
        ProcessQueue& operator= (const ProcessQueue&);// nicht implementiert
    public:
                    ProcessQueue (  Model&  owner,
                            const   String& name,
                                    bool    showInReport = true,
                                    bool    showInTrace  = true);
                    ProcessQueue (const ProcessQueue&);
        virtual    ~ProcessQueue ();

                void        Insert (Process& p);
                void        InsertAfter (Process& p, Process& where);
                void        InsertBefore (Process& p, Process& where);
                void        Remove (Process& p);

                Process&    First () const;
                Process&    Last () const;
                Process&    Pred (const Process& p) const;
                Process&    Succ (const Process& p) const;

                Process&    First (Condition& c) const;
                Process&    Last (Condition& c) const;
                Process&    Pred (const Process& p, Condition& c) const;
                Process&    Succ (const Process& p, Condition& c) const;

                String      ClassName () const;

    private:
        QueueImpl&          qimpl;
};

// --------------------------------------------------------------------------

#endif  // PROCESSQUEUE_H
```

pqueue.cc

```
// --------------------------------------------------------------------------
//
// Datei
//          pqueue.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// --------------------------------------------------------------------------

#include "pqueue.h"

#include "process.h"
#include "msgqueue.h"
#include "qimpl.h"
#include "repqueue.h"

#include <assert.h>

// --------------------------------------------------------------------------

static const char* className = "ProcessQueue";

// --------------------------------------------------------------------------
```

```
ProcessQueue::ProcessQueue (Model& owner, const String& name,
                            bool showInReport, bool showInTrace)
    :   QueueBased  (owner, name, showInReport, showInTrace),
        qimpl       (*new QueueImpl (*this))
    {}

// ---------------------------------------------------------------------------
// Statistikdaten werden kopiert
// aktuelle Laenge ist 0, da eine leere Warteschlange entsteht

ProcessQueue::ProcessQueue (const ProcessQueue& q)
    :   QueueBased  (q),        // leere Warteschlange erzeugen
        qimpl       (*new QueueImpl (*this))
    {}

// ---------------------------------------------------------------------------

ProcessQueue::~ProcessQueue ()
    {
        delete &qimpl;
    }

// ---------------------------------------------------------------------------

void ProcessQueue::Insert (Process& p)
    {
        const char* where = "ProcessQueue::Insert";

        if (!valid (className, where))
            return;
        if (qimpl.Insert (p, where))
            // Trace
            if (TraceIsOn())
                SendMessage (TrcQInsert (*this, p));
    }

// ---------------------------------------------------------------------------

void ProcessQueue::InsertBefore (Process& p, Process& before)
    {
        const char* where = "ProcessQueue::InsertBefore";

        if (!valid (className, where))
            return;
        if (qimpl.InsertBefore (p, before, where))
            // Trace
            if (TraceIsOn())
                SendMessage (TrcQInsertAt (*this, p, before, true));
    }

// ---------------------------------------------------------------------------

void ProcessQueue::InsertAfter (Process& p, Process& after)
    {
        const char* where ="ProcessQueue::InsertAfter";

        if (!valid (className, where))
            return;
        if (qimpl.InsertAfter (p, after, where))
            // Trace
            if (TraceIsOn())
                SendMessage (TrcQInsertAt (*this, p, after, false));
    }

// ---------------------------------------------------------------------------

void ProcessQueue::Remove (Process& p)
{
    const char* where = "ProcessQueue::Remove";

    if (!valid (className, where))
        return;
    if (!qimpl.Remove (p, where))
        SendMessage (QMsgRemoveNotFound (where, *this, p));
    else
        // Trace
        if (TraceIsOn())
            SendMessage (TrcQRemove (*this, p));
}

// ---------------------------------------------------------------------------

Process& ProcessQueue::First() const
```

```
    {
        const char* where = "ProcessQueue::First";

        if (!valid (className, where))
            return NullProcess();

        return (Process&)qimpl.First (where);
    }

// ---------------------------------------------------------------------------

Process& ProcessQueue::First (Condition& c) const
    {
        const char* where = "ProcessQueue::First";

        if (!valid (className, where))
            return NullProcess();

        Process& p = (Process&)qimpl.First (c, where);
        if (!p.IsNullProcess() && TraceIsOn())
            SendMessage (TrcQFind (*this, p));
        return p;
    }

// ---------------------------------------------------------------------------

Process& ProcessQueue::Last() const
    {
        const char* where = "ProcessQueue::Last";

        if (!valid (className, where))
            return NullProcess();

        return (Process&)qimpl.Last (where);
    }

// ---------------------------------------------------------------------------

Process& ProcessQueue::Last (Condition& c) const
    {
        const char* where = "ProcessQueue::Last";

        if (!valid (className, where))
            return NullProcess();

        Process& p = (Process&)qimpl.Last (c, where);
        if (!p.IsNullProcess() && TraceIsOn())
            SendMessage (TrcQFind (*this, p));
        return p;
    }

// ---------------------------------------------------------------------------

Process& ProcessQueue::Pred (const Process& p) const
    {
        const char* where = "ProcessQueue::Pred";

        if (!valid (className, where))
            return NullProcess();

        return (Process&)qimpl.Pred (p, where);
    }

// ---------------------------------------------------------------------------

Process& ProcessQueue::Pred (const Process& p, Condition& c) const
    {
        const char* where = "ProcessQueue::Pred";

        if (!valid (className, where))
            return NullProcess();

        Process& p2 = (Process&)qimpl.Pred (p, c, where);
        if (!p2.IsNullProcess() && TraceIsOn())
            SendMessage (TrcQFind (*this, p2));
        return p2;
    }

// ---------------------------------------------------------------------------

Process& ProcessQueue::Succ (const Process& p) const
    {
        const char* where = "ProcessQueue::Succ";
```

```
        if (!valid (className, where))
            return NullProcess();

        return (Process&)qimpl.Succ (p, where);
    }

// ----------------------------------------------------------------------------

Process& ProcessQueue::Succ (const Process& p, Condition& c) const
    {
        const char* where = "ProcessQueue::Succ";

        if (!valid (className, where))
            return NullProcess();

        Process& p2 = (Process&)qimpl.Succ (p, c, where);
        if (!p2.IsNullProcess() && TraceIsOn())
            SendMessage (TrcQFind (*this, p2));
        return p2;
    }

// ----------------------------------------------------------------------------

String ProcessQueue::ClassName () const
    {
        return className;
    }

// ----------------------------------------------------------------------------
```

process.h

```
// ----------------------------------------------------------------------------
//
// Datei
//          process.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ----------------------------------------------------------------------------

#ifndef PROCESS_H
#define PROCESS_H

// ----------------------------------------------------------------------------

#include "entity.h"

#include "interrup.h"

#include "boolean.h"
#include "simtime.h"
#include "str.h"

// ----------------------------------------------------------------------------

class Res;
class ProcessQueue;

class ProcessImplementation;

// ----------------------------------------------------------------------------

class Process : public Entity
    /*  Process dient als Oberklasse fuer alle aktiven dynamischen
        Simulationsobjekte. Prozesse sind Entities, d.h. sie koennen in
        der erieignisorientierten Modellierung auch als solche behandelt
        werden. */
```

```
    /*  Sie unterscheiden sich von Entities dadurch, dass sie ein eigenes
        Verhalten aufweisen. Dieses Verhalten muss in der rein virtuellen
        Methode LifeCycle () beschrieben werden. Ist die Methode vollstaendig
        abgearbeitet, so gilt der Prozess als terminiert, er kann nicht mehr
        aktiviert werden.
    */
{
        friend class    ProcessImplementation;  // Die Implementierung der
                                                // internen Aulaufsteuerung
        friend class    ProcessCooperation;     // muss Aufrufe durchreichen
        friend class    Scheduler;              // muss Zugriff auf die
                                                // Koroutine haben
        friend class    WaitQueue;              // Zugriff auf Cooperate,
                                                // blocked und waitQueue
        friend class    ProcessBlocker;         // Zugriff auf blocked

                    Process (const Process&);   // nicht implementiert
        Process&    operator= (const Process&); // nicht implementiert

    public:
                    Process (Model& owner, const String& name = "",
                             bool showInTrace = true);
        virtual    ~Process ();

                bool            IsNullProcess () const;
                                    // ist dieses der Pseudo-Prozess?

                bool            Terminated () const;
                bool            Blocked () const;

                void            Activate (SimTime dt);
                                    /*  Vormerken zum Zeitpunkt now + dt */
                void            ReActivate (SimTime dt);
                                    /*  analog zu ReSchedule */
                void            ActivateBefore (Schedulable& before);

                void            ActivateAfter (Schedulable& after);

                void            Interrupt (const InterruptCode& reason);
                InterruptCode   GetInterruptCode() const;
                bool            Interrupted() const;
                void            ClearInterruptCode ();

                Process&        Master () const;
                bool            CanCooperate () const;

                void            Cooperate (ProcessCooperation& coop);
                    /*  Der laufende Prozess (Master) ruft Cooperate am Slave
                        auf, um mit ihm die in coop beschriebenen Handlungen
                        auszufuehren. Nach der Kooperation werden erst der
                        Master dann der Slave aktiviert. */

                bool            ReleasedAll (Res*& r, unsigned long& n);
                    /*  true, wenn alle Ressourcen zurueckgegeben wurden
                        dann bleibt r unveraendert und n wird auf 0 gesetzt.
                        Andernfalls wird in r eine Ressource zurueckgegeben,
                        von der der Prozess noch n Einheiten belegt. */

                String          ClassName () const;

    protected:
        virtual void            LifeCycle () = 0;
                    /* Handlungen des Prozesses */

                void            Hold (SimTime dt);
                    /*  Reaktivierung bei now + dt. Konzeptionell hat der
                        Prozess hier seine aktive zeitverbrauchende Phase. */

                void            Passivate ();
                    /*  Passivierung fuer unbestimmte Zeit. Der Prozess kann
                        nur noch durch andere Objekte wieder aktiviert werden.
                    */

    private:
                void            Start();        // Die erste Aktivierung
                void            resetMaster();  // master auf 0,
                                                // Interrupt kopieren

                ProcessImplementation&  coroutine;
                InterruptCode           interruptCode;
                bool                    terminated;
                bool                    blocked;  // falls blockiert
                Process*                master;   // falls P. als Slave koop.
                ProcessQueue*           waitQueue;// falls P. als Slave wartet
```

```
};

// ---------------------------------------------------------------------------

#endif  // PROCESS_H
```

process.cc

```
// ---------------------------------------------------------------------------
//
// Datei
//          process.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#include "process.h"

#include "coroutin.h"
#include "experimm.h"
#include "model.h"
#include "pimpl.h"
#include "resdb.h"
#include "schedule.h"
#include "coop.h"
#include "pqueue.h"
#include "msgsched.h"

#include <assert.h>

// ---------------------------------------------------------------------------

static const char* className = "Process";

// ---------------------------------------------------------------------------

Process::Process (Model& owner, const String& name, bool trace)
    :   Entity              (owner, name, trace),
        coroutine           (*new ProcessImplementation (*this)),
        interruptCode       (InterruptCode::NoInterrupt()),
        terminated          (false),
        blocked             (false),
        master              (0),
        waitQueue           (0)
    {
        isProcess = true;
    }

// ---------------------------------------------------------------------------

Process::~Process ()
    {
        const char* where = "Process::~Process()";

        if (*this == CurrentProcess() && !IsNullProcess())
        {   // Fehler: der laufende Prozess kann nicht geloescht werden
            FatalError ("the current process must not be deleted!", where);
            assert(false);
        }

        ExperimentManager::Instance().GetResourceDB (*this).
            Destroy (this, GetModel().InDestruction());
        delete &coroutine;
    }

// ---------------------------------------------------------------------------
```

```
void Process::Start()
    {
        LifeCycle();
        terminated = true;
        if (TraceIsOn())
            SendMessage (TrcTerminate());
        if (CheckDeleteOnTermination())
            ExperimentManager::Instance().GetScheduler (*this).
                Terminate (*this);
        coroutine.MainCoroutine()->Transfer();  // Kontrolle an den Scheduler
    }

// ---------------------------------------------------------------------------

void Process::Activate (SimTime dt)
    {
        const char* where = "Process::Activate";

        // this pruefen
        if (!valid (className, where))
            return;
        if (IsScheduled())
        {
            Warning (   "attempt to activate an already activated process",
                        where, "ignored");
            return;
        }
        if (Terminated())
        {
            Warning (   "attempt to activate a terminated process",
                        where, "ignored");
            return;
        }
        if (Blocked())
        {
            Warning (   "attempt to activate a blocked process",
                        where, "ignored");
            return;
        }

        if (dt < 0.0 && dt != NOW())
        {
            Warning (   "negativ dt [" + String(dt.Time()) + ']',
                        where, "0.0 is used");
            dt = 0.0;
        }

        if (TraceIsOn())
            SendMessage (TrcActivate(dt, *this));
        resetMaster();
        Scheduler& scheduler = ExperimentManager::Instance().
                                   GetScheduler(*this);
        scheduler.Schedule (dt, *this);
    }

// ---------------------------------------------------------------------------

void Process::ReActivate (SimTime dt)
    {
        const char* where = "Process::ReActivate";

        if (!valid (className, where))
            return;
        if (!IsScheduled())
        {
            Warning (   "attempt to reactivate an unactivated process",
                        where, "ignored");
            return;
        }
        if (Terminated())
        {
            Warning (   "attempt to reactivate a terminated process",
                        where, "ignored");
            return;
        }
        if (Blocked())
        {
            Warning (   "attempt to activate a blocked process",
                        where, "ignored");
            return;
        }

        if (dt < 0.0 && dt != NOW())
        {
```

```
            Warning (   "negativ dt [" + String(dt.Time()) + ']',
                        where, "0.0 is used");
            dt = 0.0;
        }

        if (TraceIsOn())
            SendMessage (TrcReActivate (dt, *this));
        resetMaster();
        Scheduler& scheduler = ExperimentManager::Instance().
                                   GetScheduler(*this);
        scheduler.ReSchedule (dt, *this);
    }

// ----------------------------------------------------------------------------

void Process::ActivateAfter (Schedulable& after)
    {
        const char* where = "Process::ActivateAfter";

        if (!valid (className, where))
            return;
        if (IsScheduled())
        {
            Warning (   "process is already scheduled", where, "ignored");
            return;
        }
        if (Terminated())
        {
            Warning (   "attempt to activate a terminated process",
                        where, "ignored");
            return;
        }
        if (Blocked())
        {
            Warning (   "attempt to activate a blocked process",
                        where, "ignored");
            return;
        }


        if (!after.Valid())
        {
            Warning (   "attempt to activate a process after an invalid one",
                        where, "ignored");
            return;
        }
        if (!IsExperimentCompatible (after))
        {
            Warning (   "attempt to activate a process after one of "
                        "another experiment",
                        where, "ignored");
            return;
        }
        if (!after.IsScheduled() && !after.IsCurrent())
        {
            Warning (   "attempt to activate a process after an "
                        "unscheduled object",
                        where, "ignored");
            return;
        }

        if (TraceIsOn())
            SendMessage (TrcActivateAfter (after, *this));
        resetMaster();

        Scheduler& scheduler = ExperimentManager::Instance().
                                   GetScheduler (*this);
        scheduler.ScheduleAfter (after, NullEvent(), *this);
    }

// ----------------------------------------------------------------------------

void Process::ActivateBefore (Schedulable& before)
    {
        const char* where = "Process::ActivateBefore";

        if (!valid (className, where))
            return;
        if (IsScheduled())
        {
            Warning (   "process is already scheduled", where);
            return;
        }
        if (Terminated())
```

```
        {
            Warning (   "attempt to activate a terminated process",
                        where, "ignored");
            return;
        }
        if (Blocked())
        {
            Warning (   "attempt to activate a blocked process",
                        where, "ignored");
            return;
        }

        if (!before.Valid())
        {
            Warning (   "attempt to activate a process before an "
                        "invalid one",
                        where, "ignored");
            return;
        }
        if (!IsExperimentCompatible (before))
        {
            Warning (   "attempt to activate a process before one of "
                        "another experiment",
                        where, "ignored");
            return;
        }
        if (before.IsCurrent() && CurrentProcess().IsNullProcess())
        {
            Warning (   "attempt to activate a process before an event",
                        where, "ignored");
            return;
        }
        if (!before.IsScheduled() && !before.IsCurrent())
        {
            Warning (   "attempt to activate a process before an "
                        "unactivated one",
                        where, "ignored");
            return;
        }

        if (TraceIsOn())
            SendMessage (TrcActivateBefore (before, *this));
        resetMaster();

        Scheduler& scheduler = ExperimentManager::Instance().
                                    GetScheduler (*this);
        scheduler.ScheduleBefore (before, NullEvent(), *this);
    }

// ---------------------------------------------------------------------------

void Process::Hold (SimTime dt)
    {
        const char* where = "Process::Hold";

        if (!valid (className, where))
            return;

        if (this != &CurrentProcess())
        {
            Warning (   "attempt to 'hold' another but the current process",
                        where, "ignored");
            return;
        }

        if (dt == NOW())
        {
            Warning (   "'Hold (NOW())' does not make sense",
                        where, "ignored");
            return;
        }

        if (dt < 0.0)
        {
            Warning (   "negativ dt [" + String(dt.Time()) + ']',
                        where, "0.0 is used");
            dt = 0.0;
        }

        Scheduler& scheduler = ExperimentManager::Instance().
                                    GetScheduler (*this);

        if (IsScheduled())
        {
```

```
            Warning (   "'Hold' of an already activated process",
                        where,
                        "the process is canceld before");
            scheduler.Cancel (*this);
        }

        if (TraceIsOn())
            SendMessage (TrcHold (dt));
        scheduler.Schedule (dt, *this);
        scheduler.Passivate (*this);
    }

// ----------------------------------------------------------------------------

void Process::Passivate()
    {
        const char* where = "Process::Passivate";

        if (!valid (className, where))
            return;

        if (*this != CurrentProcess())
        {
            Warning (   "attempt to 'passivate' another but the "
                        "current process",
                        where, "ignored");
            return;
        }

        if (TraceIsOn())
            SendMessage (TrcPassivate ());

        Scheduler& scheduler = ExperimentManager::Instance().
                                   GetScheduler (*this);
        scheduler.Passivate (*this);
    }

// ----------------------------------------------------------------------------
// ----------------------------------------------------------------------------

void Process::Interrupt (const InterruptCode& ic)
    {
        const char* where = "Process::Interrupt";

        if (!valid (className, where))
            return;

        if (master)
            if (master->Valid())
            {   // this kooperiert als Slave => Master unterbrechen
                if (TraceIsOn())
                    SendMessage (TrcInterruptSlave (*this));
                master->Interrupt (ic);
                return;
            }
            else
                master = 0;

        if (Terminated())
        {
            Warning (   "attempt to interrupt a terminated process",
                        where, "ignored");
            return;
        }
        if (Blocked())
        {
            Warning (   "attempt to interrupt a blocked process",
                        where, "will be ignored");
            return;
        }

        if (interruptCode != InterruptCode::NoInterrupt())
            Warning (   "attempt to interrupt a process with a set "
                        "interruptCode",
                        where, "code will be overwritten",
                        "you should call 'ClearInterruptCode' before");

        if (TraceIsOn())
            SendMessage (TrcInterrupt (*this, ic));
        interruptCode = ic;

        if (*this != CurrentProcess())
        {
            if (IsScheduled())
```

```
            {
                SkipTraceNote(2);
                Cancel();
            }
            else
                SkipTraceNote();
            ActivateAfter (Current());
        }
    }

// ---------------------------------------------------------------------------

InterruptCode Process::GetInterruptCode () const
    {
        const char* where = "Process::GetInterruptCode";

        if (!valid (className, where))
            return InterruptCode::NoInterrupt();

        return interruptCode;
    }

// ---------------------------------------------------------------------------

bool Process::Interrupted () const
    {
        const char* where = "Process::Interrupted";

        if (!valid (className, where))
            return false;

        return interruptCode != InterruptCode::NoInterrupt();
    }

// ---------------------------------------------------------------------------

void Process::ClearInterruptCode ()
    {
        const char* where = "Process::ClearInterruptCode";

        if (!valid (className, where))
            return;

        interruptCode = InterruptCode::NoInterrupt();
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

Process& Process::Master() const
    {
        const char* where = "Process::Master";

        if (valid (className, where))
            if (master && valid (*master, className, where))
                return *master;

        return NullProcess();
    }

// ---------------------------------------------------------------------------

bool Process::CanCooperate() const
    {
        const char* where = "Process::CanCooperate";

        if (valid (className, where))
            return master == 0;

        return false;
    }

// ---------------------------------------------------------------------------

bool Process::Terminated() const
    {
        const char* where = "Process::Terminated";

        if (valid (className, where))
            return terminated;

        return true;
    }
```

```
// --------------------------------------------------------------------------

bool Process::Blocked() const
    {
        const char* where = "Process::Blocked";

        if (valid (className, where))
            return blocked;

        return false;
    }

// --------------------------------------------------------------------------

String Process::ClassName () const
    {
        return className;
    }

// --------------------------------------------------------------------------

bool Process::IsNullProcess () const
    {
        return this == &NullProcess();
    }

// --------------------------------------------------------------------------

bool Process::ReleasedAll (Res*& res, unsigned long& n)
    {
        const char* where = "Process::ReleasedAll";
        if (!valid (className, where))
            return false;

        ResourceDB& rdb = ExperimentManager::Instance().GetResourceDB (*this);
        return (n = rdb.AskProvider (res, this)) > 0;
    }

// --------------------------------------------------------------------------

void Process::Cooperate (ProcessCooperation& coop)
    {
        // this ist der Slave, current der Master

        const char* where = "Process::Cooperate";

        // slave pruefen
        if (!valid (className, where))
            return;
        if (master)
        {
            Warning (   "slaves can cooperate only with one master at a time",
                        where);
            return;
        }
        if (terminated)
        {
            Warning (   "slave is already terminated",
                        where);
            return;
        }

        // Kooperation pruefen
        if (!valid (coop, "ProcessCooperation", where))
            return;
        if (!IsExperimentCompatible (coop))
        {
            Warning ("attemp to mix components of different experiments",
                     where, "ignored");
            return;
        }
        if (!IsModelCompatible (coop))
        {
            Warning (   "incompatible " + coop.ClassName() + ' '
                      + coop.QuotedName(),
                        where, "ignored");
            return;
        }

        // master pruefen
        Process& current = CurrentProcess();
        if (!valid (current, className, where))
            return;
        if (current.IsNullProcess())
```

```
        {
            Warning (   "only processes can cooperate with other processes",
                        where, "ignored");
            return;
        }
        if (!IsExperimentCompatible (current))
        {
            Warning ("attemp to mix components of different experiments",
                     where, "ignored");
            return;
        }
        if (!IsModelCompatible (current))
        {
            Warning (   "incompatible " + current.ClassName() + ' '
                      + current.QuotedName(),
                        where, "ignored");
            return;
        }

        // slave muss in einer WaitQueue warten
        if (!waitQueue)
        {
            Warning (   "slaves must wait in a WaitQueue before it "
                        "can be cooperated with",
                        where, "ignored");
            return;
        }

        if (!valid ((ModelComponent&)*waitQueue, "WaitQueue", where))
            return;

        master = &current;
        if (master->TraceIsOn())
            SendMessage (TrcCooperate (*this, *waitQueue, coop));

        // slave aus der WaitQueue entfernen
        waitQueue->Remove (*this);
        waitQueue = 0;
        blocked = false;

        coop.Cooperation (*master, *this);

        if (coop.CheckDeleteOnTermination())
        {
            Scheduler& scheduler = ExperimentManager::Instance().
                                       GetScheduler (*this);
            scheduler.Terminate (coop);
        }

        if (Valid())
        {
            assert (&current == &CurrentProcess());
            if (master && (master == &current))
                // master ist beim Slave nur gleich current, wenn er weder
                // bereits aktiviert, noch bereits mit einem anderen master
                // kooperiert.
                // Unterbrechungsursache von Master auf Slave kopieren
                // das geschieht in den Activate-Methoden ueber 'resetMaster'
                ActivateAfter (Current());  // setzt master auf 0
        }
    }

// ------------------------------------------------------------------------------

void Process::resetMaster ()
    {
        if (master && master->Valid ())
            interruptCode = master->interruptCode;
        master = 0;
    }

// ------------------------------------------------------------------------------
```

qbased.h

```
// ------------------------------------------------------------------------------
//
// Datei
```

```
//          qbased.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// -----------------------------------------------------------------------------

#ifndef QUEUEBASED_H
#define QUEUEBASED_H

// -----------------------------------------------------------------------------

#include "reportab.h"   // Basisklasse

#include "boolean.h"
#include "simtime.h"
#include "str.h"

// -----------------------------------------------------------------------------

#define Undefined           double(-1)

// -----------------------------------------------------------------------------

class QueueBased : public Reportable
    /*  fuehrt die Statistik von Queues.
    */
{
        friend class QueueImpl;     // hat Zugriff auf Statistik
        QueueBased& operator=  (const QueueBased&);
                        // Zuweisung nicht implementiert
    public:
                    QueueBased (Model&  owner,
                        const   String& name,
                                bool    showInReport = true,
                                bool    showInTrace  = true);
                    QueueBased (const QueueBased&); // Kopierkonstruktor
        virtual    ~QueueBased ();

        virtual void            Reset();
        virtual Reporter*       NewReporter() const;

                bool            Empty () const;
                unsigned long   Length () const;

                unsigned long   MinLength () const;
                unsigned long   MaxLength () const;
                double          AvgLength () const;
                double          StdDevLength () const;
                SimTime         MinLengthAt () const;
                SimTime         MaxLengthAt () const;

                unsigned long   ZeroWaits () const;

                SimTime         MaxWaitTime () const;
                SimTime         AvgWaitTime () const;
                SimTime         StdDevWaitTime () const;
                SimTime         MaxWaitTimeAt () const;

                String          ClassName () const;
    private:
                void            updateStatistics ();
                void            RemoveWithWarning (Entity&      e,
                                                   const char*  where);
                // die folgenden Methoden werden QueueImpl aufgerufen,
                // um die Statistik zu aktualisieren.
                void            addItem ();
                void            delItem (const SimTime& timeIn);

        // Statistik
        unsigned long   length; // aktuelle Laenge
        unsigned long   minLength, maxLength,
                        zeroWaits;
        double          wSumLength, wSumSquareLength;
        SimTime         lastAccess,
```

```
                        minLengthAt, maxLengthAt,
                        maxWaitTime, maxWaitTimeAt,
                        sumWaitTime, sumSquareWaitTime;
};

// -----------------------------------------------------------------------------

#endif  // QUEUEBASED_H
```

qbased.cc

```
// -----------------------------------------------------------------------------
//
// Datei
//          qbased.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// -----------------------------------------------------------------------------

#include "qbased.h"

#include "model.h"
#include "process.h"
#include "repqueue.h"

#include <assert.h>
#include <math.h>

// -----------------------------------------------------------------------------

static const char* className = "QueueBased";

// -----------------------------------------------------------------------------

QueueBased::QueueBased (Model&  owner,
                const   String& name,
                        bool    showInReport,
                        bool    showInTrace)
    :   Reportable  (owner, name, showInReport, showInTrace),
        length      (0),
        minLength   (0),    maxLength        (0),
        zeroWaits   (0),
        wSumLength  (0),    wSumSquareLength (0),
        lastAccess  (0),
        minLengthAt (0),    maxLengthAt      (0),
        maxWaitTime (0),    maxWaitTimeAt    (0),
        sumWaitTime (0),    sumSquareWaitTime(0)
    {}

// -----------------------------------------------------------------------------
// muss implementiert sein, da der Konstruktor von Queue private ist

QueueBased::QueueBased (const QueueBased& qb)
    :   Reportable  (qb),
        length      (0),            // leere Warteschlange erzeugen
        minLength   (0),             maxLength        (qb.maxLength),
        zeroWaits   (qb.zeroWaits),
        wSumLength  (qb.wSumLength), wSumSquareLength (qb.wSumSquareLength),
        lastAccess  (qb.lastAccess),
        minLengthAt (CurrentTime()), maxLengthAt      (qb.maxLengthAt),
        maxWaitTime (qb.maxWaitTime),maxWaitTimeAt    (qb.maxWaitTimeAt),
        sumWaitTime (qb.sumWaitTime),sumSquareWaitTime(qb.sumSquareWaitTime)
    {}

// -----------------------------------------------------------------------------

QueueBased::~QueueBased ()
```

```
    {}

// --------------------------------------------------------------------------

void QueueBased::Reset()
    {
        // Reset baseClass
        Reportable::Reset();    // gibt Warnung wenn ungueltig
        if (!Valid()) return;   // nur noch abbrechen

        minLength           =
        maxLength           = Length();
        zeroWaits           = 0;
        wSumLength          =
        wSumSquareLength    = 0;
        sumWaitTime         =
        sumSquareWaitTime   =
        maxWaitTime         = 0;
        maxWaitTimeAt       =
        minLengthAt         =
        maxLengthAt         =
        lastAccess          = CurrentTime();
    }

// --------------------------------------------------------------------------

Reporter* QueueBased::NewReporter() const
    {
        return new QueueReporter (*this);
    }

// --------------------------------------------------------------------------

bool QueueBased::Empty() const
    {
        return length <= 0;
    }

// --------------------------------------------------------------------------

unsigned long QueueBased::Length() const
    {
        return length;
    }

// --------------------------------------------------------------------------

unsigned long QueueBased::MinLength() const
    {
        return minLength;
    }

// --------------------------------------------------------------------------

unsigned long QueueBased::MaxLength() const
    {
        return maxLength;
    }

// --------------------------------------------------------------------------

double QueueBased::AvgLength() const
    {
        const char* where = "QueueBased::AvgLength";

        if (!valid (className, where))
            return Undefined;

        SimTime  now  = CurrentTime();
        SimTime  diff = now - ResetAt();

        if (diff < Epsilon())
            return Undefined;
        else
            // aktualisierte Werte beruecksichtigen
            return (wSumLength + Length() * (now - lastAccess).Time())
                / diff.Time();
    }

// --------------------------------------------------------------------------

double QueueBased::StdDevLength(void) const
    {
        const char* where = "QueueBased::StdDevLength";
```

```
        if (!valid (className, where))
            return Undefined;

        SimTime  now  = CurrentTime();
        SimTime  diff = now - ResetAt();

        if (diff < Epsilon())
            return Undefined;

        // aktualisierte Werte beruecksichtigen
        double  len     = double(Length()),
                mean    = AvgLength();
        SimTime span = now - lastAccess;

        return sqrt(fabs((wSumSquareLength + len * len * span.Time())
            / diff.Time() - mean * mean));
    }

// ------------------------------------------------------------------------------

SimTime QueueBased::MinLengthAt() const
    {
        return minLengthAt;
    }

// ------------------------------------------------------------------------------

SimTime QueueBased::MaxLengthAt() const
    {
        return maxLengthAt;
    }

// ------------------------------------------------------------------------------

unsigned long QueueBased::ZeroWaits() const
    {
        return zeroWaits;
    }

// ------------------------------------------------------------------------------

SimTime QueueBased::MaxWaitTime() const
    {
        return maxWaitTime;
    }

// ------------------------------------------------------------------------------

SimTime QueueBased::AvgWaitTime() const
    {
        if (Observations() > 0)
            return sumWaitTime / double(Observations());
        else
            return Undefined;
    }

// ------------------------------------------------------------------------------

SimTime QueueBased::StdDevWaitTime() const
    {
        if (Observations() > 0) {
            double  mean = AvgWaitTime().Time();
            double  dobs = double(Observations());

            // um einen Ueberlauf beim Produkt zweier CARDINALs
            // weitgehend zu verhindern, werden die Operanden
            // getrennt umgewandelt!

            return sqrt(fabs(dobs * sumSquareWaitTime.Time() - mean * mean)
                / dobs * (dobs - 1.0));
        } else
            return Undefined;
    }

// ------------------------------------------------------------------------------

SimTime QueueBased::MaxWaitTimeAt() const
    {
        return maxWaitTimeAt;
    }

// ------------------------------------------------------------------------------
```

```
String QueueBased::ClassName () const
    {
        return className;
    }

// ---------------------------------------------------------------------------

void QueueBased::addItem()
    {
        updateStatistics();
        length++;

        if (length > maxLength)
        {
            maxLength   = length;
            maxLengthAt = CurrentTime();
        }
    }

// ---------------------------------------------------------------------------

void QueueBased::delItem(const SimTime& timeIn)
    {
        updateStatistics();

        SimTime currentTime = CurrentTime();
        SimTime waitTime    = currentTime - timeIn;

        sumWaitTime         += waitTime;
        sumSquareWaitTime   += (waitTime * waitTime).Time();

        if (maxWaitTime < waitTime) {
            maxWaitTime     = waitTime;
            maxWaitTimeAt   = currentTime;
        }
        if (waitTime < Epsilon())
            zeroWaits++;

        assert (length > 0);
        length--;
        if (length < minLength)
        {
            minLength   = length;
            minLengthAt = currentTime;
        }
        IncObservations();
    }

// ---------------------------------------------------------------------------

void QueueBased::updateStatistics()
    {
        SimTime         now     = CurrentTime();
        SimTime         diff    = now - lastAccess;
        unsigned long   len     = Length();

        wSumLength       += len * diff.Time();
        wSumSquareLength += len * len * diff.Time();
        lastAccess        = now;
    }

// ---------------------------------------------------------------------------
```

qimpl.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          qimpl.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
```

```
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#ifndef QUEUEIMPL_H
#define QUEUEIMPL_H

// ---------------------------------------------------------------------------

#include "str.h"

// ---------------------------------------------------------------------------

class Entity;
class QueueLink;
class QueueBased;
class Condition;

// ---------------------------------------------------------------------------

class QueueImpl
    /*  QueueImpl implementiert die Warteschlangenfunktionalitaet.
    */
{       friend class QueueLink; // muss Zugriff auf 'RemoveWithWarning' haben

                QueueImpl (const QueueImpl&);           // nicht implementiert
                QueueImpl& operator= (const QueueImpl&);// nicht implementiert
    public:
                QueueImpl (QueueBased& qb);
                            // qb ist das Objekt, das die Statistik fuehrt
               ~QueueImpl ();

                bool        Insert      (Entity& e, const char* where = 0);
                bool        InsertAfter (Entity& e, Entity& after,
                                         const char* where = 0);
                bool        InsertBefore(Entity& e, Entity& before,
                                         const char* where = 0);
                bool        Remove      (Entity& e, const char* where = 0);

                Entity&     First (const char* where = 0) const;
                Entity&     Last (const char* where = 0) const;
                Entity&     Pred (const Entity& e,
                                  const char* where = 0) const;
                Entity&     Succ (const Entity& e,
                                  const char* where = 0) const;

                Entity&     First (Condition& c,
                                   const char* where = 0) const;
                                // DESMO: Find()
                Entity&     Last (Condition& c,
                                  const char* where = 0) const;
                                // beginnt die Suche am Ende
                Entity&     Pred (const Entity& e, Condition& c,
                                  const char* where = 0) const;
                                // DESMO: FindNext()
                Entity&     Succ (const Entity& e, Condition& c,
                                  const char* where = 0) const;
                                // sucht rueckwaerts

                String      Name () const;
        virtual String      Debug() const;

    private:
        void                RemoveWithWarning (Entity& e,
                                               const char* where = 0);
        QueueLink*          getQueueLink   (const Entity&,
                                            const char* where) const;
        bool                CheckInsert    (const Entity&,
                                            const char* where) const;
        bool                CheckInsertAt  (const Entity& valid,
                                            const Entity& invalid,
                                            const char* at,
                                            const char* where) const;
        bool                CheckRemove    (const Entity&,
                                            const char* where) const;
        bool                CheckPredSucc  (const Entity&,
                                            bool  predOrSucc,
                                            const char* where) const;
        bool                CheckCondition (const Condition&,
                                            const char* where) const;
        // Verbindung zu Entities
        QueueLink       *first,
```

```
                       *last;
        QueueBased&     qBased;
};

// -----------------------------------------------------------------------------

#endif  // QUEUEIMPL_H
```

qimpl.cc

```
// -----------------------------------------------------------------------------
//
// Datei
//          qimpl.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// -----------------------------------------------------------------------------

#include "qimpl.h"

#include "conditio.h"
#include "entity.h"
#include "expopts.h"
#include "msgqueue.h"
#include "qbased.h"
#include "qlink.h"
#include "repqueue.h"
#include "text.h"

#include <assert.h>
#include <iostream.h>
#include <iomanip.h>
#include <math.h>

// -----------------------------------------------------------------------------

static const char* className = "QueueBased";

// -----------------------------------------------------------------------------

QueueImpl::QueueImpl (QueueBased& queueBased)
    :   first   (0),
        last    (0),
        qBased  (queueBased)
    {}

// -----------------------------------------------------------------------------

QueueImpl::~QueueImpl ()
    {
        while (first) {
            QueueLink* qLink = first;
            first = qLink->Next();
            delete qLink;
        }
    }

// -----------------------------------------------------------------------------

bool QueueImpl::Insert (Entity& e, const char* w)
    {
        const char* where = ((w) ? w : "QueueImpl::Insert");

        if (!qBased.valid (className, where))
            return false;
        if (!CheckInsert (e, where))
            return false;
```

```
        QueueLink* qLink = new QueueLink(*this, e, qBased.CurrentTime(),
                                         where);

        qBased.addItem();

        if (!first)
            first = last = qLink;
        else {
            QueueLink* tmp = last;

            do {
                if (e <= tmp->GetObject())
                    break;
            } while ((tmp = tmp->Prev()) != 0);

            if (tmp)
                qLink->InsertBehind (*tmp);
            else
                qLink->InsertBefore (*first);

            if (first->Prev()) first = first->Prev();
            if (last->Next()) last = last->Next();
        }

        // Debug
        if (qBased.DebugIsOn())
            qBased.SendMessage (DebugMessage(Debug()));

        return true;
    }

// ---------------------------------------------------------------------------

bool QueueImpl::InsertBefore (Entity& e, Entity& before, const char* w)
    {
        const char* where = ((w) ? w : "QueueImpl::InsertBefore");

        if (!qBased.valid (className, where))
            return false;
        if (!CheckInsert (e, where))
            return false;
        if (!CheckInsertAt (e, before, txtBefore, where))
            return false;

        QueueLink* tmp = getQueueLink (before, where);

        if (!tmp) {
            qBased.SendMessage (QMsgInsertAtNotFound (where,  qBased, e,
                                                      before, txtBefore));
            return false;
        }

        QueueLink* qLink = new QueueLink (*this, e, qBased.CurrentTime(),
                                          where);

        qBased.addItem();
        assert (first != 0);
        qLink->InsertBefore (*tmp);
        if (first->Prev()) first = first->Prev();

        // Debug
        if (qBased.DebugIsOn())
            qBased.SendMessage (DebugMessage(Debug()));
        return true;
    }

// ---------------------------------------------------------------------------

bool QueueImpl::InsertAfter (Entity& e, Entity& after, const char* w)
    {
        const char* where = ((w) ? w : "QueueImpl::InsertAfter");

        if (!qBased.valid (className, where))
            return false;
        if (!CheckInsert (e, where))
            return false;
        if (!CheckInsertAt (e, after, txtAfter, where))
            return false;

        QueueLink* tmp = getQueueLink (after, where);

        if (!tmp) {
            qBased.SendMessage (QMsgInsertAtNotFound (where, qBased, e,
```

```
                                                       after, txtAfter));
            return false;
        }

        QueueLink* qLink = new QueueLink (*this, e, qBased.CurrentTime(),
                                          where);

        qBased.addItem();
        assert (last);
        qLink->InsertBehind (*tmp);
        if (last->Next()) last = last->Next();

        // Debug
        if (qBased.DebugIsOn())
            qBased.SendMessage (DebugMessage(Debug()));
        return true;
    }

// ------------------------------------------------------------------------------

void QueueImpl::RemoveWithWarning (Entity& e, const char* w)
{
    const char* where = ((w) ? w : "QueueImpl::RemoveWithWarning");

    if (!qBased.valid (className, where))
        return;

    qBased.Warning ("Attempt to insert " + e.QuotedName() +
                    " (which can be only in one queue at a time) into "
                    + qBased.QuotedName(),
                    where,
                    e.QuotedName() + " will be removed from " +
                    qBased.QuotedName());
    //qBased.SkipTraceNote();
    Remove (e);
}

// ------------------------------------------------------------------------------

bool QueueImpl::Remove (Entity& e, const char* w)
{
    const char* where = ((w) ? w : "QueueImpl::Remove");

    if (!qBased.valid (className, where))
        return false;
    if (!CheckRemove (e, where))
        return false;

    QueueLink* qLink = first;

    while (qLink)
    {
        if (&qLink->GetObject() == &e)
        {
            if (qLink == first) first = qLink->Next();
            if (qLink == last)  last  = qLink->Prev();

            qBased.delItem (qLink->TimeIn());

            delete qLink;

            // Debug
            if (qBased.DebugIsOn())
                qBased.SendMessage (DebugMessage(Debug()));

            return true;
        }
        qLink = qLink->Next();
    }
    qBased.SendMessage (QMsgRemoveNotFound (where, qBased, e));
    return false;
}

// ------------------------------------------------------------------------------

Entity& QueueImpl::First(const char* w) const
    {
        const char* where = ((w) ? w : "QueueImpl::First");

        if (!qBased.valid (className, where))
            return qBased.NullEntity();

        if (!first)
            return qBased.NullEntity();
```

```
        return first->GetObject();
    }

// ----------------------------------------------------------------------------

Entity& QueueImpl::First (Condition& c, const char* w) const
    {
        const char* where = ((w) ? w : "QueueImpl::First");

        if (!qBased.valid (className, where))
            return qBased.NullEntity();
        if (!CheckCondition (c, where))
            return qBased.NullEntity();

        QueueLink* qLink = first;
        while (qLink)
        {
            if (c.Check (qLink->GetObject()))
                return qLink->GetObject ();
            else
                qLink = qLink->Next();
        }
        return qBased.NullEntity();
    }

// ----------------------------------------------------------------------------

Entity& QueueImpl::Last(const char* w) const
    {
        const char* where = ((w) ? w : "QueueImpl::Last");

        if (!qBased.valid (className, where))
            return qBased.NullEntity();
        if (!last)
            return qBased.NullEntity();
        return last->GetObject();
    }

// ----------------------------------------------------------------------------

Entity& QueueImpl::Last (Condition& c, const char* w) const
    {
        const char* where = ((w) ? w : "QueueImpl::Last");

        if (!qBased.valid (className, where))
            return qBased.NullEntity();
        if (!CheckCondition (c, where))
            return qBased.NullEntity();

        QueueLink* qLink = last;
        while (qLink)
        {
            if (c.Check (qLink->GetObject()))
                return qLink->GetObject ();
            else
                qLink = qLink->Prev();
        }
        return qBased.NullEntity();
    }

// ----------------------------------------------------------------------------

Entity& QueueImpl::Pred (const Entity& e, const char* w) const
    {
        const char* where = ((w) ? w : "QueueImpl::Pred");

        if (!qBased.valid (className, where))
            return qBased.NullEntity();
        if (!CheckPredSucc (e, true, where))
            return qBased.NullEntity();

        QueueLink*      qLink = getQueueLink (e, where);

        if (qLink && qLink->Prev())
            return qLink->Prev()->GetObject();
        return qBased.NullEntity();
    }

// ----------------------------------------------------------------------------

Entity& QueueImpl::Pred (const Entity& e, Condition& c, const char* w) const
    {
        const char* where = ((w) ? w : "QueueImpl::Pred");
```

```
        if (!qBased.valid (className, where))
            return qBased.NullEntity();
        if (!CheckPredSucc (e, true, where))
            return qBased.NullEntity();
        if (!CheckCondition (c, where))
            return qBased.NullEntity();

        QueueLink*      qLink = getQueueLink (e, where);

        if (qLink)
        {
            qLink = qLink->Prev();
            while (qLink)
            {
                if (c.Check (qLink->GetObject()))
                {
                    if (qBased.TraceIsOn())
                        qBased.SendMessage (TrcQFind (qBased,
                                                      qLink->GetObject()));
                    return qLink->GetObject ();
                } else
                    qLink = qLink->Prev();
            }
        }
        return qBased.NullEntity();
    }

// ----------------------------------------------------------------------------

Entity& QueueImpl::Succ (const Entity& e, const char* w) const
    {
        const char* where = ((w) ? w : "QueueImpl::Succ");

        if (!qBased.valid (className, where))
            return qBased.NullEntity();
        if (!CheckPredSucc (e, false, where))
            return qBased.NullEntity();

        QueueLink* qLink = getQueueLink (e, where);

        if (qLink && qLink->Next())
            return qLink->Next()->GetObject();
        return qBased.NullEntity();
    }

// ----------------------------------------------------------------------------

Entity& QueueImpl::Succ (const Entity& e, Condition& c, const char* w) const
    {
        const char* where = ((w) ? w : "QueueImpl::Succ");

        if (!qBased.valid (className, where))
            return qBased.NullEntity();
        if (!CheckPredSucc (e, false, where))
            return qBased.NullEntity();
        if (!CheckCondition (c, where))
            return qBased.NullEntity();

        QueueLink* qLink = getQueueLink (e, where);

        if (qLink)
        {
            qLink = qLink->Next();
            while (qLink)
            {
                if (c.Check (qLink->GetObject()))
                {
                    if (qBased.TraceIsOn())
                        qBased.SendMessage (TrcQFind (qBased,
                                                      qLink->GetObject()));
                    return qLink->GetObject ();
                } else
                    qLink = qLink->Next();
            }
        }
        return qBased.NullEntity();
    }

// ----------------------------------------------------------------------------

QueueLink* QueueImpl::getQueueLink (const Entity& e, const char* where) const
    {
       QueueLink* tmp = QueueLink::GetQueueLink (e);
```

```
        while (tmp && &tmp->GetQueue() != this)
            tmp = tmp->Same();

        if (!tmp)
            qBased.Warning ("Unable to find entity " + e.QuotedName()
                         + " in queue " + qBased.QuotedName(),
                         where);
        return tmp;
    }

// ---------------------------------------------------------------------------
// Check-Routinen

bool QueueImpl::CheckInsert (const Entity& e, const char* where) const
    {
        if (!e.Valid())
        {
            qBased.SendMessage (QMsgInsertInvalid(where, qBased));
            return false;
        }
        if (e.IsNullEntity())
        {
            qBased.SendMessage (QMsgInsertNullEntity(where, qBased));
            return false;
        }
        if (!qBased.IsExperimentCompatible (e))
        {
            qBased.Warning ("attemp to mix components of different "
                            "experiments", where, "ignored");
            return false;
        }
        if (!qBased.IsModelCompatible (e))
        {
            qBased.SendMessage (QMsgInsertIncompatible(where, qBased, e));
            return false;
        }
        return true;
    }

// ---------------------------------------------------------------------------

bool QueueImpl::CheckInsertAt (const Entity& e1, const Entity& e2,
                               const char* at, const char* where) const
    {
        if (!e2.Valid())
        {
            qBased.SendMessage (QMsgInsertAtInvalid(where, qBased, e1, at));
            return false;
        }
        if (e2.IsNullEntity())
        {
            qBased.SendMessage (QMsgInsertAtNullEntity(where, qBased,
                                                       e1, at));
            return false;
        }
        if (!qBased.IsExperimentCompatible (e2))
        {
            qBased.Warning ("attemp to mix components of different "
                            "experiments", where, "ignored");
            return false;
        }
        if (!qBased.IsModelCompatible (e2))
        {
            qBased.SendMessage (QMsgInsertAtIncompatible(where, qBased,
                                                         e1, e2, at));
            return false;
        }
        return true;
    }

// ---------------------------------------------------------------------------

bool QueueImpl::CheckRemove (const Entity& e, const char* where) const
    {
        if (!e.Valid())
        {
            qBased.SendMessage (QMsgRemoveInvalid(where, qBased));
            return false;
        }
        if (e.IsNullEntity())
        {
            qBased.SendMessage (QMsgRemoveNullEntity(where, qBased));
            return false;
        }
```

```
        if (!qBased.IsExperimentCompatible (e))
        {
            qBased.Warning ("attemp to mix components of different "
                            "experiments", where, "ignored");
            return false;
        }
        if (!qBased.IsModelCompatible (e))
        {
            qBased.SendMessage (QMsgRemoveIncompatible(where, qBased, e));
            return false;
        }
        return true;
    }

// ------------------------------------------------------------------------

bool QueueImpl::CheckCondition (const Condition& c, const char* where) const
    {
        if (!qBased.valid (c, "Condition", where))
            return false;
        if (!qBased.IsExperimentCompatible (c))
        {
            qBased.Warning ("attemp to mix components of different "
                            "experiments", where, "ignored");
            return false;
        }
        if (!qBased.IsModelCompatible (c))
        {
            qBased.SendMessage (QMsgIncompatibleCondition(where, qBased, c));
            return false;
        }
        return true;
    }

// ------------------------------------------------------------------------

bool QueueImpl::CheckPredSucc (const Entity& e, bool predOrSucc,
                               const char* where) const
    {
        if (!e.Valid())
        {
            qBased.SendMessage (QMsgPredSuccInvalid
                                    (where, qBased, predOrSucc));
            return false;
        }
        if (e.IsNullEntity())
        {
            qBased.SendMessage (QMsgPredSuccNullEntity
                                    (where, qBased, predOrSucc));
            return false;
        }
        if (!qBased.IsExperimentCompatible (e))
        {
            qBased.Warning ("attemp to mix components of different "
                            "experiments", where, "ignored");
            return false;
        }
        if (!qBased.IsModelCompatible (e))
        {
            qBased.SendMessage (QMsgPredSuccIncompatible
                                    (where, qBased, e, predOrSucc));
            return false;
        }
        return true;
    }

// ------------------------------------------------------------------------

String QueueImpl::Name () const
    {
        return qBased.Name();
    }

// ------------------------------------------------------------------------

String QueueImpl::Debug () const
    {
        strstream  ss;
        long       oflgs = ss.flags(ios::showpoint | ios::fixed | ios::right);
        int        tw = qBased.GetExperimentOpts().TimeWidth();
        int        nw = qBased.GetExperimentOpts().NameWidth();
        int        pw = 8;
        int        w  = nw + pw + tw +2;
```

```
        ss  << "Entities waiting in " << qBased.QuotedName()
            << " at ClockTime : " << qBased.CurrentTime() << endl
            << resetiosflags(ios::left) << setfill('-')
            << setw (w) << "" << setfill (' ') << endl
            << setiosflags(ios::left) << setw (nw) << "Entity" << ' '
            << resetiosflags(ios::left) << setw (pw) << "Priority" << ' '
            << resetiosflags(ios::left) << setw (tw) << "Entry in q" << endl
            << resetiosflags(ios::left) << setfill('-')
            << setw (w) << "" << setfill (' ') << endl
            << resetiosflags(ios::left);

        for (QueueLink* qLink = first; qLink; qLink = qLink->Next())
        {
            ss  << setiosflags(ios::left) << setw(nw)
                << qLink->GetObject().Name().Left(nw)
                << resetiosflags(ios::left) << setw(pw+1)
                << qLink->GetObject().GetPriority()
                << setw(tw+1) << qLink->TimeIn() << endl;
        }
        ss << ends;

        return qBased.Debug() + String(ss);
    }

// ---------------------------------------------------------------------------
```

qlink.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          qlink.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------
//
// Weiterentwicklung von:
//
// Diplomarbeit
//
//          Entwurf und Realisierung eines objektorientierten
//          Simulationspakets in C++
//
// Author
//          Heiko Weber
//
// Beschreibung
//
//          Die Klasse QueueLink stellt die Funktionalitaet zum Einfuegen
//          eines Objektes in eine Queue zur Verfuegung.
//
// ---------------------------------------------------------------------------

#ifndef  QUEUELINK_H
#define  QUEUELINK_H

// ---------------------------------------------------------------------------

#include "simtime.h"

// ---------------------------------------------------------------------------

class QueueImpl;
class Entity;

// ---------------------------------------------------------------------------
```

```
class QueueLink
{
                    QueueLink (const QueueLink&);   // nicht definiert!

    public:
                    QueueLink (QueueImpl&, Entity&, const SimTime&,
                                                    const char* where = "");
        virtual     ~QueueLink();

                QueueLink&      InsertBefore (QueueLink&);
                QueueLink&      InsertBehind (QueueLink&);
                Entity&         GetObject() const;
        static  QueueLink*      GetQueueLink (const Entity&);
                QueueImpl&      GetQueue() const;
                QueueLink*      Prev() const;
                QueueLink*      Next() const;
                QueueLink*      Same() const;
        const   SimTime&        TimeIn() const;

    private:
        QueueLink&          remove();

        QueueImpl&          queue;  // die Warteschlange
        Entity&             object; // das wartende Entity
        QueueLink*          prev,   // Links zu anderen Eintraegen
                            *next,
                            *same;  // same Entity in same or other queue
        SimTime             timeIn;
};

// ------------------------------------------------------------------------

#endif //QUEUELINK_H
```

qlink.cc

```
// ------------------------------------------------------------------------
//
// Datei
//          qlink.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ------------------------------------------------------------------------
//
// Weiterentwicklung von:
//
// Diplomarbeit
//
//          Entwurf und Realisierung eines objektorientierten
//          Simulationspakets in C++
//
// Author
//          Heiko Weber
//
// Beschreibung
//
//          Die Klasse QueueLink stellt die Funktionalitaet zum Einfuegen
//          eines Objektes in eine Queue zur Verfuegung.
//
// ------------------------------------------------------------------------

#include <assert.h>
#include "qlink.h"
#include "entity.h"
#include "qimpl.h"

// ------------------------------------------------------------------------
```

```
QueueLink::QueueLink (QueueImpl& q, Entity& o,  const SimTime& t,
                                                const char* where)
    :   queue(q), object(o),
        prev(0), next(0), same(o.qlink),
        timeIn(t)
    {
        o.qlink = this;

        switch (object.GetQueueOption()) {
            case OnlyOneQueue:
                while (same)
                    same->GetQueue().RemoveWithWarning (object, where);
                break;
            case MultipleQueue:
                {
                    QueueLink* ql = same, *tmp;

                    while (ql)
                        if (&ql->GetQueue() == &queue) {
                            tmp = ql;
                            ql  = ql->same;
                            tmp->GetQueue().Remove (object);
                        } else
                            ql = ql->same;
                }
        }
    }

// ------------------------------------------------------------------------------

QueueLink::~QueueLink()
    {
        remove();
    }

// ------------------------------------------------------------------------------

QueueLink& QueueLink::InsertBefore (QueueLink& ql)
{
    assert (prev == 0 && next == 0);

    prev    = ql.prev;
    next    = &ql;
    if (prev)
        prev->next = this;
    ql.prev = this;

    return *this;
}

// ------------------------------------------------------------------------------

QueueLink& QueueLink::InsertBehind (QueueLink& ql)
    {
        assert (prev == 0 && next == 0);

        next    = ql.next;
        if (next)
            next->prev = this;
        prev    = &ql;
        ql.next = this;

        return *this;
    }

// ------------------------------------------------------------------------------

Entity& QueueLink::GetObject() const
    {
        return object;
    }

// ------------------------------------------------------------------------------

QueueLink* QueueLink::GetQueueLink (const Entity& e)
    {
        return e.qlink;
    }

// ------------------------------------------------------------------------------

QueueImpl& QueueLink::GetQueue() const
    {
```

```
        return queue;
    }

// ---------------------------------------------------------------------------

QueueLink* QueueLink::Prev() const
    {
        return prev;
    }

// ---------------------------------------------------------------------------

QueueLink* QueueLink::Next() const
    {
        return next;
    }

// ---------------------------------------------------------------------------

QueueLink* QueueLink::Same() const
    {
        return same;
    }

// ---------------------------------------------------------------------------

const SimTime& QueueLink::TimeIn() const
    {
        return timeIn;
    }

// ---------------------------------------------------------------------------

QueueLink& QueueLink::remove()
    {
     /* alte fehlerhafte Version
        if (prev) prev->next = next;
        if (next) next->prev = prev;
        object.qlink = same;

        next = prev = same = 0;
        return *this;
     */

        // korrigierte Version (TS)
        if (prev) prev->next = next;
        if (next) next->prev = prev;

        if (object.qlink == this)
            // this ist der Anfang der Same-Kette
            // object auf Nachfolger in der Same-Kette zeigen lassen
            object.qlink = same;
        else
        {   // object.qlink bleibt unberuehrt
            // aber evtl. Luecke in der Same-Kette schliessen
            QueueLink*  ql = object.qlink;
            while (ql && ql->same != this)
                ql = ql->same;
            if (ql && ql->same == this)
                    // (andernfalls gab es nur ein ql zu diesem object)
                ql->same = same;    // Luecke geschlossen
        }

        next = prev = same = 0;
        return *this;
    }

// ---------------------------------------------------------------------------
```

queue.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          queue.h
//
// Diplomarbeit
//
```

```
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#ifndef QUEUE_H
#define QUEUE_H

// ---------------------------------------------------------------------------

#include "qbased.h"     // Basisklasse

#include "conditio.h"   // Condition
#include "str.h"

// ---------------------------------------------------------------------------

class Entity;
class QueueLink;

// ---------------------------------------------------------------------------

class Queue : public QueueBased
    /*  Queue ist die Klasse der Warteschlangen, in denen Entities
        warten koenen.
    */
{
    public:
                Queue (const Queue&);   // Kopierkonstruktor
                Queue (Model& owner, const String& name = "",
                       bool   showInReport = true,
                       bool   showInTrace  = true);
               ~Queue ();

                void            Insert      (Entity& e);
                void            InsertAfter (Entity& e, Entity& where);
                void            InsertBefore(Entity& e, Entity& where);
                void            Remove      (Entity& e);

                Entity&         First () const;
                Entity&         Last () const;
                Entity&         Pred (const Entity& e) const;
                Entity&         Succ (const Entity& e) const;

                Entity&         First (Condition& c) const;
                                    // DESMO: Find()
                Entity&         Last (Condition& c) const;
                                    // beginnt die Suche am Ende
                Entity&         Pred (const Entity& e, Condition& c) const;
                                    // DESMO: FindNext()
                Entity&         Succ (const Entity& e, Condition& c) const;
                                    // sucht rueckwaerts
                String          ClassName () const;

    private:
        QueueImpl&              qimpl;
};

// ---------------------------------------------------------------------------

#endif  // QUEUE_H
```

queue.cc

```
// ---------------------------------------------------------------------------
//
// Datei
//          queue.cc
//
// Diplomarbeit
//
```

```
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// -----------------------------------------------------------------------------

#include "queue.h"

#include "entity.h"
#include "msgqueue.h"
#include "qimpl.h"
#include "repqueue.h"

#include <assert.h>

// -----------------------------------------------------------------------------

static const char* className = "Queue";

// -----------------------------------------------------------------------------

Queue::Queue (  Model& owner, const String& name,
                bool showInReport, bool showInTrace)
    :   QueueBased  (owner, name, showInReport, showInTrace),
        qimpl       (*new QueueImpl (*this))
    {}

// -----------------------------------------------------------------------------
// Kopierkonstruktor: Statistikdaten werden kopiert
// aktuelle Laenge ist 0, da eine leere Warteschlange entsteht

Queue::Queue (const Queue& q)
    :   QueueBased  (q),        // leere Warteschlange erzeugen
        qimpl       (*new QueueImpl (*this))
    {}

// -----------------------------------------------------------------------------

Queue::~Queue ()
    {
        delete &qimpl;
    }

// -----------------------------------------------------------------------------

void Queue::Insert (Entity& e)
    {
        const char* where = "Queue::Insert";

        if (!valid (className, where))
            return;
        if (qimpl.Insert (e, where))
            // Trace
            if (TraceIsOn())
                SendMessage (TrcQInsert (*this, e));
    }

// -----------------------------------------------------------------------------

void Queue::InsertBefore (Entity& e, Entity& before)
    {
        const char* where = "Queue::InsertBefore";

        if (!valid (className, where))
            return;
        if (qimpl.InsertBefore (e, before, where))
            // Trace
            if (TraceIsOn())
                SendMessage (TrcQInsertAt (*this, e, before, true));
    }

// -----------------------------------------------------------------------------

void Queue::InsertAfter (Entity& e, Entity& after)
    {
        const char* where ="Queue::InsertAfter";

        if (!valid (className, where))
```

```
            return;
        if (qimpl.InsertAfter (e, after, where))
            // Trace
            if (TraceIsOn())
                SendMessage (TrcQInsertAt (*this, e, after, false));
    }

// ----------------------------------------------------------------------------

void Queue::Remove (Entity& e)
{
    const char* where = "Queue::Remove";

    if (!valid (className, where))
        return;
    if (!qimpl.Remove (e, where))
        SendMessage (QMsgRemoveNotFound (where, *this, e));
    else
        // Trace
        if (TraceIsOn())
            SendMessage (TrcQRemove (*this, e));
}

// ----------------------------------------------------------------------------

Entity& Queue::First() const
    {
        const char* where = "Queue::First";

        if (!valid (className, where))
            return NullEntity();

        return qimpl.First (where);
    }

// ----------------------------------------------------------------------------

Entity& Queue::First (Condition& c) const
    {
        const char* where = "Queue::First";

        if (!valid (className, where))
            return NullEntity();

        Entity& e = qimpl.First (c, where);
        if (!e.IsNullEntity() && TraceIsOn())
            SendMessage (TrcQFind (*this, e));
        return e;
    }

// ----------------------------------------------------------------------------

Entity& Queue::Last() const
    {
        const char* where = "Queue::Last";

        if (!valid (className, where))
            return NullEntity();

        return qimpl.Last (where);
    }

// ----------------------------------------------------------------------------

Entity& Queue::Last (Condition& c) const
    {
        const char* where = "Queue::Last";

        if (!valid (className, where))
            return NullEntity();

        Entity& e = qimpl.Last (c, where);
        if (!e.IsNullEntity() && TraceIsOn())
            SendMessage (TrcQFind (*this, e));
        return e;
    }

// ----------------------------------------------------------------------------

Entity& Queue::Pred (const Entity& e) const
    {
        const char* where = "Queue::Pred";

        if (!valid (className, where))
```

```
            return NullEntity();

        return qimpl.Pred (e, where);
    }

// ---------------------------------------------------------------------------

Entity& Queue::Pred (const Entity& e, Condition& c) const
    {
        const char* where = "Queue::Pred";

        if (!valid (className, where))
            return NullEntity();

        Entity& e2 = qimpl.Pred (e, c, where);
        if (!e2.IsNullEntity() && TraceIsOn())
            SendMessage (TrcQFind (*this, e2));
        return e2;
    }

// ---------------------------------------------------------------------------

Entity& Queue::Succ (const Entity& e) const
    {
        const char* where = "Queue::Succ";

        if (!valid (className, where))
            return NullEntity();

        return qimpl.Succ (e, where);
    }

// ---------------------------------------------------------------------------

Entity& Queue::Succ (const Entity& e, Condition& c) const
    {
        const char* where = "Queue::Succ";

        if (!valid (className, where))
            return NullEntity();

        Entity& e2 = qimpl.Succ (e, c, where);
        if (!e2.IsNullEntity() && TraceIsOn())
            SendMessage (TrcQFind (*this, e2));
        return e2;
    }

// ---------------------------------------------------------------------------

String Queue::ClassName () const
    {
        return className;
    }

// ---------------------------------------------------------------------------
```

realdist.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          realdist.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#ifndef REALDIST_H
```

```
#define REALDIST_H

// ---------------------------------------------------------------------------

#include "distribu.h"   // Basisklasse

#include "str.h"

// ---------------------------------------------------------------------------

class RealDist : public Distribution
{
    public:
        virtual double      Sample () = 0;

        virtual    ~RealDist ();
    protected:
                    RealDist (  Model& owner,
                                const String& name = "",
                                bool showInReport = true,
                                bool showInTrace = false);

        void    swap (double &a, double &b) { double t = a; a = b; b = t; }
};

// ---------------------------------------------------------------------------

class RealDistConst : public RealDist
{
    public:
                    RealDistConst ( Model& owner,
                                    const String& name = "",
                                    double value = 0.0,
                                    bool showInReport = true,
                                    bool showInTrace = false);
        virtual    ~RealDistConst ();

        virtual double      Sample ();
        virtual String      GetType() const;
                                //  liefert die Typ-Bezeichnung des ZZ-Stroms
                double      GetValue() const;
                void        ChangeParameter (double newValue);
        virtual Reporter*   NewReporter() const;
    private:
                double      value;
};

// ---------------------------------------------------------------------------

class RealDistUniform : public RealDist
{
    public:
                    RealDistUniform (   Model& owner,
                                        const String& name = "",
                                        double low  = 0.0,
                                        double high = 0.0,
                                        bool showInReport = true,
                                        bool showInTrace = false);
        virtual    ~RealDistUniform ();

        virtual double      Sample ();
        virtual String      GetType() const;
                                //  liefert die Typ-Bezeichnung des ZZ-Stroms
                double      GetLow() const;
                double      GetHigh() const;
                void        ChangeParameter (double newLow, double newHigh);
        virtual Reporter*   NewReporter() const;
    protected:
                void        checkHiLo (const char* where);
    private:
                double      low, high;
};

// ---------------------------------------------------------------------------

class RealEmpiricalEntry;

class RealDistEmpirical : public RealDist
{
    public:
                    RealDistEmpirical ( Model& owner,
                                        const String& name = "",
                                        bool showInReport = true,
                                        bool showInTrace = false);
```

```
                    RealDistEmpirical (const RealDistEmpirical& objToCopy);
        virtual    ~RealDistEmpirical ();

        virtual double      Sample ();
        virtual String      GetType() const;
                                //  liefert die Typ-Bezeichnung des ZZ-Stroms
                void        AddEntry (double newValue,
                                      double cumulativeFrequency);
                unsigned    CountEntries () const;
                double      GetValue (unsigned entry) const;
                                // 0 <= entry < CountEntries()
                double      GetCumulativeFrequency (unsigned entry) const;
                                // 0 <= entry < CountEntries()
        virtual Reporter*   NewReporter() const;
    private:
        unsigned            entries;
        RealEmpiricalEntry* table;
};

// ----------------------------------------------------------------------------

class RealDistExponential : public RealDist
{
    public:
                    RealDistExponential (   Model& owner,
                                            const String& name = "",
                                            double mean = 0.0,
                                            bool showInReport = true,
                                            bool showInTrace = false);
        virtual    ~RealDistExponential ();

        virtual double      Sample ();
        virtual String      GetType() const;
                                //  liefert die Typ-Bezeichnung des ZZ-Stroms
                double      GetMean() const;
                void        ChangeParameter (double newMean);
        virtual Reporter*   NewReporter() const;
    protected:
                void        checkMean (const char* where);
    private:
                double      mean;
};

// ----------------------------------------------------------------------------

class RealDistErlang : public RealDist
{
    public:
                    RealDistErlang (    Model& owner,
                                        const String& name = "",
                                        unsigned k = 1,
                                        double mean = 0.0,
                                        bool showInReport = true,
                                        bool showInTrace = false);
        virtual    ~RealDistErlang ();

        virtual double      Sample ();
        virtual String      GetType() const;
                                //  liefert die Typ-Bezeichnung des ZZ-Stroms
                unsigned    GetK() const;
                double      GetMean() const;
                void        ChangeParameter (unsigned newK, double newMean);
        virtual Reporter*   NewReporter() const;
    protected:
                void        checkK (const char* where);
                void        checkMean (const char* where);
    private:
                unsigned    k;
                double      mean;
};

// ----------------------------------------------------------------------------

class RealDistNormal : public RealDist
{
    public:
                    RealDistNormal (    Model& owner,
                                        const String& name = "",
                                        double mean     = 0.0,
                                        double stddev   = 0.0,
                                        bool showInReport = true,
                                        bool showInTrace = false);
        virtual    ~RealDistNormal ();
```

```
        virtual double      Sample ();
        virtual String      GetType() const;
                                //  liefert die Typ-Bezeichnung des ZZ-Stroms
                double      GetMean() const;
                double      GetStddev() const;
                void        ChangeParameter (double mean, double stddev);
        virtual Reporter*   NewReporter() const;
    protected:
                void        checkStddev (const char* where);
    private:
                double      mean, stddev;
                double      u, v;
                bool        evenSample;
};

// ---------------------------------------------------------------------------

#endif  // REALDIST_H
```

realdist.cc

```
// ---------------------------------------------------------------------------
//
// Datei
//          realdist.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#include "realdist.h"

#include <assert.h>
#include <math.h>
#include "msgdist.h"
#include "repdist.h"

// ---------------------------------------------------------------------------

static const char* className = "RealDist";

// ---------------------------------------------------------------------------

RealDist::RealDist (    Model& owner,
                        const String& name,
                        bool showInReport,
                        bool showInTrace)
    :   Distribution(owner, name, showInReport, showInTrace)
    {}

// ---------------------------------------------------------------------------

RealDist::~RealDist ()
    {};

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

RealDistConst::RealDistConst (  Model& owner,
                                const String& name,
                                double Value,
                                bool showInReport,
                                bool showInTrace)
    :   RealDist(owner, name, showInReport, showInTrace),
        value(Value)
    {
        state = Distribution::Initialized;
    }
```

```
// ---------------------------------------------------------------------------

RealDistConst::~RealDistConst ()
    {}

// ---------------------------------------------------------------------------

double RealDistConst::Sample ()
    {
        const char* where = "RealDistConst::Sample";
        if (!valid (className, where))
            return -1;
        checkSample (where);
        IncObservations();
        if (TraceIsOn())
            SendMessage (TrcDistRSample (*this, value));
        return value;
    }

// ---------------------------------------------------------------------------

String RealDistConst::GetType () const
    {
        return "R-Constant";
    }

// ---------------------------------------------------------------------------

double RealDistConst::GetValue () const
    {
        return value;
    }

// ---------------------------------------------------------------------------

void RealDistConst::ChangeParameter (double newValue)
    {
        const char* where = "RealDistConst::ChangeParameter";
        if (checkParam (where))
            value = newValue;
    }

// ---------------------------------------------------------------------------

Reporter* RealDistConst::NewReporter () const
    {
        return new RealDistConstReporter (*this);
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

RealDistUniform::RealDistUniform (  Model& owner,
                                    const String& name,
                                    double Low,
                                    double High,
                                    bool showInReport,
                                    bool showInTrace)
    :   RealDist(owner, name, showInReport, showInTrace),
        low(Low),
        high(High)
    {
        const char* where = "RealDistUniform::RealDistUniform";
        state = Distribution::Initialized;
        checkHiLo (where);
    }

// ---------------------------------------------------------------------------

RealDistUniform::~RealDistUniform ()
    {}

// ---------------------------------------------------------------------------

void RealDistUniform::checkHiLo (const char* where)
    {
        if (high < low)
        {
            swap (high, low);
            SendMessage (MsgDistUnifSwap (where, *this, low, high));
        }
    }
```

```
// ----------------------------------------------------------------------------

double RealDistUniform::Sample ()
    {
        const char* where = "RealDistUniform::Sample";
        if (!valid (className, where))
            return -1;
        checkSample (where);
        IncObservations();

        double r = low + ((high - low) * random());
        if (TraceIsOn())
            SendMessage (TrcDistRSample (*this, r));
        return r;
    }

// ----------------------------------------------------------------------------

String RealDistUniform::GetType () const
    {
        return "R-Uniform";
    }

// ----------------------------------------------------------------------------

double RealDistUniform::GetLow () const
    {
        return low;
    }

// ----------------------------------------------------------------------------

double RealDistUniform::GetHigh () const
    {
        return high;
    }

// ----------------------------------------------------------------------------

void RealDistUniform::ChangeParameter (double newLow, double newHigh)
    {
        const char* where = "RealDistUniform::ChangeParameter";
        if (checkParam (where))
        {
            low     = newLow;
            high    = newHigh;
            checkHiLo (where);
        }
    }

// ----------------------------------------------------------------------------

Reporter* RealDistUniform::NewReporter () const
    {
        return new RealDistUniformReporter (*this);
    }

// ----------------------------------------------------------------------------
// ----------------------------------------------------------------------------

struct  RealEmpiricalEntry
{
    double              value;
    double              probability;

                        RealEmpiricalEntry(double v = 0, double p = 0)
                            : value(v), probability(p) {}
};

// ----------------------------------------------------------------------------

RealDistEmpirical::RealDistEmpirical (  Model& owner,
                                        const String& name,
                                        bool showInReport,
                                        bool showInTrace)
    :   RealDist(owner, name, showInReport, showInTrace),
        entries(0),
        table(0)
    {}

// ----------------------------------------------------------------------------

RealDistEmpirical::RealDistEmpirical (const RealDistEmpirical& rde)
    :   RealDist    (rde),
```

```
        entries     (rde.entries),
        table       (new RealEmpiricalEntry [entries])
    {
        for (int i = 0; i < entries; i++)
            table [i] = rde.table [i];
    }

// ------------------------------------------------------------------------

RealDistEmpirical::~RealDistEmpirical ()
    {
        delete[] table;
        table = 0;
        entries = 0;
    }

// ------------------------------------------------------------------------

String RealDistEmpirical::GetType () const
    {
        return "R-Empirical";
    }

// ------------------------------------------------------------------------

double RealDistEmpirical::Sample ()
    {
        const char* where = "RealDistEmpirical::Sample";
        if (!valid (className, where))
            return -1;
        checkSample (where);

        double   q = random();
        unsigned i = 1;

        assert (entries); //ERROR
        while (i < entries && table[i].probability < q)
            i++;
        IncObservations();

        double r = table[i-1].value
                    + (table[i].value       - table[i-1].value)
                    * (q                    - table[i-1].probability)
                    / (table[i].probability - table[i-1].probability);
        if (TraceIsOn())
            SendMessage (TrcDistRSample (*this, r));
        return r;
    }

// ------------------------------------------------------------------------

void RealDistEmpirical::AddEntry (double value, double probability)
    {
        const char* where = "RealDistEmpirical::AddEntry";
        if (checkParam (where))
        {
            RealEmpiricalEntry* t = new RealEmpiricalEntry [entries + 1];

            for (unsigned n = 0; n < entries; n++)
            {
                t [n] = table [n];
                if (t [n].probability >= probability)
                {
                    SendMessage (MsgDistEmpProbOrder (where, *this,
                                                      probability,
                                                      t[n].probability));
                    delete[] t;
                    return; // Error
                }
            }
            t [entries++] = RealEmpiricalEntry (value, probability);
            if (table)
                delete[] table;
            table = t;
            if (probability >= 1.0)
                state = Distribution::Initialized;
        }
    }

// ------------------------------------------------------------------------

unsigned RealDistEmpirical::CountEntries () const
    {
        return entries;
```

```
    }

// ---------------------------------------------------------------------------

double RealDistEmpirical::GetValue (unsigned entry) const
    {
        const char* where = "RealDistEmpirical::GetValue";
        if (entry < entries)
        {
            return table [entry].value;
        } else
        {
            SendMessage (MsgDistEmpWrongIndex (where, *this, entry, entries));
            if (entries)
                return table [entries -1].value;
            else
                return 0.0;
        }
    }

// ---------------------------------------------------------------------------

double RealDistEmpirical::GetCumulativeFrequency (unsigned entry) const
    {
        const char* where = "RealDistEmpirical::GetCumulativeFrequency";
        if (entry < entries)
        {
            return table [entry].probability;
        } else
        {
            SendMessage (MsgDistEmpWrongIndex (where, *this, entry, entries));
            if (entries)
                return table [entries -1].probability;
            else
                return 0.0;
        }
    }

// ---------------------------------------------------------------------------

Reporter* RealDistEmpirical::NewReporter () const
    {
        return new RealDistEmpiricalReporter (*this);
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

RealDistExponential::RealDistExponential (  Model& owner,
                                            const String& name,
                                            double Mean,
                                            bool showInReport,
                                            bool showInTrace)
    :   RealDist(owner, name, showInReport, showInTrace),
        mean(Mean)
    {
        const char* where = "RealDistExponential::RealDistExponential";
        state = Distribution::Initialized;
        checkMean (where);
    }

// ---------------------------------------------------------------------------

RealDistExponential::~RealDistExponential ()
    {}

// ---------------------------------------------------------------------------

void RealDistExponential::checkMean (const char* where)
    {
        if (mean < 0.0)
        {
            SendMessage (MsgDistMeanNeg (where, *this, mean));
            mean = -mean;
        }
    }

// ---------------------------------------------------------------------------

double RealDistExponential::Sample ()
    {
        const char* where = "RealDistExponential::Sample";
        if (!valid (className, where))
            return -1;
```

```
        checkSample (where);
        IncObservations();

        double r = -log (random()) * mean;
        if (TraceIsOn())
            SendMessage (TrcDistRSample (*this, r));
        return r;
    }

// ------------------------------------------------------------------------------

String RealDistExponential::GetType () const
    {
        return "Neg-Expon.";
    }

// ------------------------------------------------------------------------------

double RealDistExponential::GetMean () const
    {
        return mean;
    }

// ------------------------------------------------------------------------------

void RealDistExponential::ChangeParameter (double newMean)
    {
        const char* where = "RealDistExponential::ChangeParameter";
        if (checkParam (where))
        {
            mean = newMean;
            checkMean (where);
        }
    }

// ------------------------------------------------------------------------------

Reporter* RealDistExponential::NewReporter () const
    {
        return new RealDistExponentialReporter (*this);
    }

// ------------------------------------------------------------------------------
// ------------------------------------------------------------------------------

RealDistErlang::RealDistErlang (    Model& owner,
                                    const String& name,
                                    unsigned K,
                                    double Mean,
                                    bool showInReport,
                                    bool showInTrace)
    :   RealDist(owner, name, showInReport, showInTrace),
        k(K),
        mean(Mean)
    {
        const char* where = "RealDistErlang::RealDistErlang";
        state = Distribution::Initialized;
        checkK (where);
        checkMean (where);
    }

// ------------------------------------------------------------------------------

RealDistErlang::~RealDistErlang ()
    {}

// ------------------------------------------------------------------------------

void RealDistErlang::checkK (const char* where)
    {
        if (k == 0)
        {
            SendMessage (MsgDistErlZeroK (where, *this));
            k = 1;
        }
    }

// ------------------------------------------------------------------------------

void RealDistErlang::checkMean (const char* where)
    {
        if (mean < 0)
        {
            SendMessage (MsgDistMeanNeg (where, *this, mean));
```

```
            mean = -mean;
        }
    }

// ----------------------------------------------------------------------------

double RealDistErlang::Sample ()
    {
        const char* where = "RealDistErlang::Sample";
        if (!valid (className, where))
            return -1;
        checkSample (where);

        double q = 1.0;

        for (unsigned n = 1; n <= k; n++)
            q *= random();
        IncObservations();

        double r = -log(q) * mean / double(k);
        if (TraceIsOn())
            SendMessage (TrcDistRSample (*this, r));
        return r;
    }

// ----------------------------------------------------------------------------

String RealDistErlang::GetType () const
    {
        return "k-Erlang";
    }

// ----------------------------------------------------------------------------

unsigned RealDistErlang::GetK () const
    {
        return k;
    }

// ----------------------------------------------------------------------------

double RealDistErlang::GetMean () const
    {
        return mean;
    }

// ----------------------------------------------------------------------------

void RealDistErlang::ChangeParameter (unsigned newK, double newMean)
    {
        const char* where = "RealDistErlang::ChangeParameter";
        if (checkParam (where))
        {
            k    = newK;
            mean = newMean;
            checkK (where);
            checkMean (where);
        }
    }

// ----------------------------------------------------------------------------

Reporter* RealDistErlang::NewReporter () const
    {
        return new RealDistErlangReporter (*this);
    }

// ----------------------------------------------------------------------------
// ----------------------------------------------------------------------------

RealDistNormal::RealDistNormal (    Model& owner,
                                    const String& name,
                                    double Mean,
                                    double Stddev,
                                    bool showInReport,
                                    bool showInTrace)
    :   RealDist(owner, name, showInReport, showInTrace),
        mean(Mean),
        stddev(Stddev),
        u(0),
        v(0),
        evenSample(false)
    {
        const char* where = "RealDistNormal::RealDistNormal";
```

```
        state = Distribution::Initialized;
        checkStddev (where);
    }

// ---------------------------------------------------------------------------

RealDistNormal::~RealDistNormal ()
    {}

// ---------------------------------------------------------------------------

void RealDistNormal::checkStddev (const char* where)
    {
        if (stddev < 0.0)
        {
            SendMessage (MsgDistStdDevNeg (where, *this, stddev));
            stddev = -stddev;
        }
    }

// ---------------------------------------------------------------------------

double RealDistNormal::Sample ()
    {
        const char* where = "RealDistNormal::Sample";
        if (!valid (className, where))
            return -1;
        checkSample (where);

        const double cPi = 3.14159265358979323846;
        double q;

        if (evenSample)
        {
            q = u * cos (v);
        } else
        {
            u = sqrt (-2.0 * log (random()));
            v = cPi * 2.0 * random();
            q = u * sin (v);
        }
        evenSample = !evenSample;
        IncObservations();

        double r = mean + (q * stddev);
        if (TraceIsOn())
            SendMessage (TrcDistRSample (*this, r));
        return r;
    }

// ---------------------------------------------------------------------------

String RealDistNormal::GetType () const
    {
        return "Normal";
    }

// ---------------------------------------------------------------------------

double RealDistNormal::GetMean () const
    {
        return mean;
    }

// ---------------------------------------------------------------------------

double RealDistNormal::GetStddev () const
    {
        return stddev;
    }

// ---------------------------------------------------------------------------

void RealDistNormal::ChangeParameter (double newMean, double newStddev)
    {
        const char* where = "RealDistNormal::ChangeParameter";
        if (checkParam (where))
        {
            mean    = newMean;
            stddev  = newStddev;
            checkStddev (where);
        }
    }
```

```
// --------------------------------------------------------------------------

Reporter* RealDistNormal::NewReporter () const
    {
        return new RealDistNormalReporter (*this);
    }

// --------------------------------------------------------------------------
```

regress.h

```
// --------------------------------------------------------------------------
//
// Datei
//          regress.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// --------------------------------------------------------------------------

#ifndef REGRESSION_H
#define REGRESSION_H

// --------------------------------------------------------------------------

#include "statobj.h"    // Basisklasse

#include "boolean.h"
#include "str.h"

// --------------------------------------------------------------------------

class ValueSupplier;
class Reporter;

// --------------------------------------------------------------------------

class Regression : public StatisticObject
{
    public:
                    Regression (Model&          owner,
                        const   String&         name,
                        const   String&         name_2,
                                ValueSupplier&  xvs,
                                ValueSupplier&  yvs,
                                bool            showInReport = true,
                                bool            showInTrace  = false);
                    Regression (Model&          owner,
                                ValueSupplier&  xvs,
                                ValueSupplier&  yvs,
                                bool            showInReport = true,
                                bool            showInTrace  = false);

        virtual void            Update ();
                String          Name2 () const;
                String          QuotedName2 () const;
                void            Values (double& x, double& y) const;
                double          xValue () const;
                double          yValue () const;
                double          xMean () const;
                double          yMean () const;
                double          ResStdDev () const;
                double          RegCoeff () const;
                double          Intercept () const;
                double          StdDevRegCoeff () const;
                double          CorrCoeff () const;

                bool            xConstant () const;
```

```
                bool            yConstant () const;

        virtual void            Reset ();
        virtual Reporter*       NewReporter () const;

                String          ClassName () const;
    protected:

    private:
        ValueSupplier&  xSupplier;
        ValueSupplier&  ySupplier;
        double          x,      y,
                        sx,     sy,
                        sxx,    syy,
                            sxy;
        String          name_2;
};

// -----------------------------------------------------------------------------

#endif  // REGRESSION_H
```

regress.cc

```
// -----------------------------------------------------------------------------
//
// Datei
//          regress.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// -----------------------------------------------------------------------------

#include "regress.h"

#include <math.h>
#include "repstat.h"
#include "valuesup.h"

// -----------------------------------------------------------------------------

static  const char*     className   = "Regression";
static  const double    cEpsilon    = 0.0001;

// -----------------------------------------------------------------------------

Regression::Regression (        Model&          owner,
                        const   String&         name,
                        const   String&         n2,
                                ValueSupplier&  xvs,
                                ValueSupplier&  yvs,
                                bool            showInReport,
                                bool            showInTrace)
    :   StatisticObject(owner, name, showInReport, showInTrace),
        xSupplier(xvs),     ySupplier(yvs),
        x(0.0),             y(0.0),
        sx(0.0),            sy(0.0),
        sxx(0.0),           syy(0.0),
        sxy(0.0),
        name_2(n2)
    {}

// -----------------------------------------------------------------------------

Regression::Regression (        Model&          owner,
                                ValueSupplier&  xvs,
                                ValueSupplier&  yvs,
                                bool            showInReport,
```

```
                                bool            showInTrace)
    :   StatisticObject(owner, "X", showInReport, showInTrace),
        xSupplier(xvs),     ySupplier(yvs),
        x(0.0),             y(0.0),
        sx(0.0),            sy(0.0),
        sxx(0.0),           syy(0.0),
        sxy(0.0),
        name_2("Y")
    {}

// ---------------------------------------------------------------------------

void Regression::Reset ()
    {
        StatisticObject::Reset();
        if (!Valid()) return;

        x   =   y   =
        sx  =   sy  =
        sxx =   syy =
        sxy =   0.0;
    }

// ---------------------------------------------------------------------------

void Regression::Update ()
    {
        const char* where = "Regression::Update";

        if (!valid (className, where))
            return;
        if (!valid (xSupplier, "x-ValueSupplier", where))
            return;
        if (!valid (ySupplier, "y-ValueSupplier", where))
            return;

        IncObservations();
        x = xSupplier.Value();
        y = ySupplier.Value();
        sx  += x;
        sy  += y;
        sxx += x * x;
        syy += y * y;
        sxy += x * y;
        traceUpdate();
    }

// ---------------------------------------------------------------------------

void Regression::Values (double& xv, double& yv) const
    {
        xv = x;
        yv = y;
    }

// ---------------------------------------------------------------------------

double Regression::xValue () const
    {
        return x;
    }

// ---------------------------------------------------------------------------

double Regression::yValue () const
    {
        return y;
    }

// ---------------------------------------------------------------------------

double Regression::xMean () const
    {
        const char* where = "Regression::xMean";

        if (!valid (className, where))
            return -1.0;

        long int n = Observations();
        if (n == 0)
        {
            Warning (   "insufficient data",
                        where, "-1.0 is returned");
            return -1.0;
```

```
        }
        return sx / double (n);
    }

// ------------------------------------------------------------------------------

double Regression::yMean () const
    {
        const char* where = "Regression::yMean";

        if (!valid (className, where))
            return -1.0;

        long int n = Observations();
        if (n == 0)
        {
            Warning (   "insufficient data",
                        where, "-1.0 is returned");
            return -1.0;
        }
        return sy / double (n);
    }

// ------------------------------------------------------------------------------

double Regression::ResStdDev () const
    {
        const char* where = "Regression::ResStdDev";

        long int    n = Observations();

        if (n <= 5)
        {
            Warning (   "insufficient data",
                        where, "-1.0 is returned");
            return -1.0;
        }

        double      dx = fabs (double (n) * sxx - sx * sx);
        double      dy = fabs (double (n) * syy - sy * sy);

        if (dx < cEpsilon || dy < cEpsilon)
        {
            Warning (   "degenerate data",
                        where, "-1.0 is returned");
            return -1.0;
        }

        // fabs gegenueber DESMO eingefuegt und sy statt y
        return  sqrt (fabs(syy - Intercept() * sy - RegCoeff() * sxy)
                / double (n - 2));
    }

// ------------------------------------------------------------------------------

double Regression::RegCoeff () const
    {
        const char* where = "Regression::RegCoeff";

        if (!valid (className, where))
            return -1.0;

        long int    n = Observations();

        if (n <= 5)
        {
            Warning (   "insufficient data",
                        where, "-1.0 is returned");
            return -1.0;
        }

        double      dx = fabs (double (n) * sxx - sx * sx);
        double      dy = fabs (double (n) * syy - sy * sy);

        if (dx < cEpsilon || dy < cEpsilon)
        {
            Warning (   "degenerate data",
                        where, "-1.0 is returned");
            return -1.0;
        }

        return  (double (n) * sxy - sx * sy) / dx;
    }
```

```
// ---------------------------------------------------------------------------

double Regression::Intercept () const
    {
        const char* where = "Regression::Intercept";

        if (!valid (className, where))
            return -1.0;

        long int    n = Observations();

        if (n <= 5)
        {
            Warning (   "insufficient data",
                        where, "-1.0 is returned");
            return -1.0;
        }

        double      dx = fabs (double (n) * sxx - sx * sx);
        double      dy = fabs (double (n) * syy - sy * sy);

        if (dx < cEpsilon || dy < cEpsilon)
        {
            Warning (   "degenerate data",
                        where, "-1.0 is returned");
            return -1.0;
        }

        return  (sy * sxx - sx * sxy) / dx;
    }

// ---------------------------------------------------------------------------

double Regression::StdDevRegCoeff () const
    {
        const char* where = "Regression::StdDevRegCoeff";

        if (!valid (className, where))
            return -1.0;

        long int    n = Observations();

        if (n <= 5)
        {
            Warning (   "insufficient data",
                        where, "-1.0 is returned");
            return -1.0;
        }

        double      dx = fabs (double (n) * sxx - sx * sx);
        double      dy = fabs (double (n) * syy - sy * sy);

        if (dx < cEpsilon || dy < cEpsilon)
        {
            Warning (   "degenerate data",
                        where, "-1.0 is returned");
            return -1.0;
        }

        return  double (n) * ResStdDev()
                / sqrt (double (n - 2) * dx);
    }

// ---------------------------------------------------------------------------

double Regression::CorrCoeff () const
    {
        const char* where = "Regression::CorrCoeff";

        long int    n = Observations();

        if (n <= 5)
        {
            Warning (   "degenerate data",
                        where, "-1.0 is returned");
            return -1.0;
        }

        double      dx = fabs (double (n) * sxx - sx * sx);
        double      dy = fabs (double (n) * syy - sy * sy);

        if (dx < cEpsilon || dy < cEpsilon)
        {
            Warning (   "degenerate data",
```

```
                        where, "-1.0 is returned");
            return -1.0;
        }

        double temp = (double (n) * sxy - sx * sy);
        return  sqrt (temp * temp / (dx * dy));
    }

// ---------------------------------------------------------------------------

String Regression::Name2 () const
    {
        return name_2;
    }

// ---------------------------------------------------------------------------

String Regression::QuotedName2 () const
    {
        return String("'") + name_2 + "'";
    }

// ---------------------------------------------------------------------------

bool Regression::xConstant () const
    {
        double  dx = fabs (double (Observations()) * sxx - sx * sx);
        return dx < cEpsilon;
    }

// ---------------------------------------------------------------------------

bool Regression::yConstant () const
    {
        double  dy = fabs (double (Observations()) * syy - sy * sy);
        return dy < cEpsilon;
    }

// ---------------------------------------------------------------------------

Reporter* Regression::NewReporter () const
    {
        return new RegressionReporter (*this);
    }

// ---------------------------------------------------------------------------

String Regression::ClassName () const
    {
        return className;
    }

// ---------------------------------------------------------------------------
```

repcondq.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          repcondq.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#ifndef CONDQUEUEREPORTER_H
#define CONDQUEUEREPORTER_H
```

```
// --------------------------------------------------------------------------

#include "repqueue.h"   // Basisklasse

#include "str.h"

// --------------------------------------------------------------------------

class CondQueue;

// --------------------------------------------------------------------------

class CondQueueReporter : public QueueReporter
{
    public:
                    CondQueueReporter (const CondQueue&);

        virtual unsigned    GetGroupID() const;
        virtual String      GetTitle() const;

        virtual ostream&    WriteHeader (ostream&) const;
        virtual ostream&    UnderscoreHeader (ostream&) const;
        virtual ostream&    WriteAsLine (ostream&, int reportWidth) const;
        virtual ostream&    WriteAsBlock(ostream&) const;
    private:
        static  unsigned    groupID;
};

// --------------------------------------------------------------------------

#endif  // CONDQUEUEREPORTER_H
```

repcondq.cc

```
// --------------------------------------------------------------------------
//
// Datei
//          repcondq.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// --------------------------------------------------------------------------

#include "repcondq.h"

#include "condq.h"
#include <iostream.h>
#include <iomanip.h>

// --------------------------------------------------------------------------

unsigned CondQueueReporter::groupID = NewGroupID();

// --------------------------------------------------------------------------

CondQueueReporter::CondQueueReporter (const CondQueue& qb)
    :   QueueReporter(qb)
    {}

// --------------------------------------------------------------------------

unsigned CondQueueReporter::GetGroupID() const
    {
        return groupID;
    }

// --------------------------------------------------------------------------
```

```
String CondQueueReporter::GetTitle() const
    {
        return "Cond-Queues";
    }

// --------------------------------------------------------------------------

ostream& CondQueueReporter::WriteHeader (ostream& os) const
    {
        return QueueReporter::WriteHeader (os)
            << ' ' << setw (3) << "All";
    }

// --------------------------------------------------------------------------

ostream& CondQueueReporter::UnderscoreHeader (ostream& os) const
    {
        char    ofill = os.fill();

        return QueueReporter::UnderscoreHeader (os)
            << setfill('-') << setw (4) << ""
            << setfill(ofill);
    }

// --------------------------------------------------------------------------

ostream& CondQueueReporter::WriteAsLine (ostream& os, int reportWidth) const
    {
        long    oflgs = os.flags(ios::showpoint | ios::fixed | ios::left);
        CondQueue&  queue = (CondQueue&)reportable;

        return QueueReporter::WriteAsLine (os, reportWidth)
            << ' ' << setw (3)
            << (queue.CheckAll() ? "yes" : "no")
            << setiosflags (oflgs);
    }

// --------------------------------------------------------------------------

ostream& CondQueueReporter::WriteAsBlock (ostream& os) const
    {
        CondQueue&  queue = (CondQueue&)reportable;

        return QueueReporter::WriteAsBlock (os) << endl
            << "All: " << (queue.CheckAll() ? "yes" : "no") << endl;
    }

// --------------------------------------------------------------------------
```

repdist.h

```
// --------------------------------------------------------------------------
//
// Datei
//          repdist.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// --------------------------------------------------------------------------

#ifndef DISTRIBREPORTER_H
#define DISTRIBREPORTER_H

// --------------------------------------------------------------------------

#include "reporter.h"   // Basisklasse

#include "str.h"
```

```
// ---------------------------------------------------------------------------

class Distribution;

class DistribReporter : public Reporter
{
    public:
                    DistribReporter (const Distribution&);
        virtual    ~DistribReporter ();

        virtual unsigned    GetGroupID() const;
        virtual String      GetTitle() const;

        virtual ostream&    WriteHeader (ostream&) const;
        virtual ostream&    UnderscoreHeader (ostream&) const;
        virtual ostream&    WriteAsLine (ostream&, int reportWidth) const;
        virtual ostream&    WriteAsBlock(ostream&) const;
    protected:
                ostream&    writeType  (ostream&) const;
                ostream&    writeParam (ostream&, double) const;
                ostream&    writeParam (ostream&, int) const;
                ostream&    writeSeed  (ostream&) const;

        virtual int         LineEnd() const;
        virtual ostream&    writeParamA(ostream&) const;
        virtual ostream&    writeParamB(ostream&) const;

    private:
        static  unsigned    groupID;
};

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

class BoolDistConst;

class BoolDistConstReporter : public DistribReporter
{
    public:
                    BoolDistConstReporter (const BoolDistConst&);
        virtual    ~BoolDistConstReporter ();
    protected:
        virtual ostream&    writeParamA (ostream&) const;
};

// ---------------------------------------------------------------------------

class BoolDistBernoulli;

class BoolDistBernoulliReporter : public DistribReporter
{
    public:
                    BoolDistBernoulliReporter (const BoolDistBernoulli&);
        virtual    ~BoolDistBernoulliReporter ();
    protected:
        virtual ostream&    writeParamA (ostream&) const;
};

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

class IntDistConst;

class IntDistConstReporter : public DistribReporter
{
    public:
                    IntDistConstReporter (const IntDistConst&);
        virtual    ~IntDistConstReporter ();
    protected:
        virtual ostream&    writeParamA (ostream&) const;
};

// ---------------------------------------------------------------------------

class IntDistUniform;

class IntDistUniformReporter : public DistribReporter
{
    public:
                    IntDistUniformReporter (const IntDistUniform&);
        virtual    ~IntDistUniformReporter ();
    protected:
        virtual ostream&    writeParamA (ostream&) const;
```

```
        virtual ostream&    writeParamB (ostream&) const;
};

// ---------------------------------------------------------------------------

class IntDistPoisson;

class IntDistPoissonReporter : public DistribReporter
{
    public:
                    IntDistPoissonReporter (const IntDistPoisson&);
        virtual    ~IntDistPoissonReporter ();
    protected:
        virtual ostream&    writeParamA (ostream&) const;
};

// ---------------------------------------------------------------------------

class IntDistEmpirical;

class IntDistEmpiricalReporter : public DistribReporter
{
    public:
                    IntDistEmpiricalReporter (const IntDistEmpirical&);
        virtual    ~IntDistEmpiricalReporter ();

        virtual ostream&    WriteAsLine (ostream&, int reportWidth) const;
        virtual ostream&    WriteAsBlock(ostream&) const;
    protected:
};

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

class RealDistConst;

class RealDistConstReporter : public DistribReporter
{
    public:
                    RealDistConstReporter (const RealDistConst&);
        virtual    ~RealDistConstReporter ();
    protected:
        virtual ostream&    writeParamA (ostream&) const;
};

// ---------------------------------------------------------------------------

class RealDistUniform;

class RealDistUniformReporter : public DistribReporter
{
    public:
                    RealDistUniformReporter (const RealDistUniform&);
        virtual    ~RealDistUniformReporter ();
    protected:
        virtual ostream&    writeParamA (ostream&) const;
        virtual ostream&    writeParamB (ostream&) const;
};

// ---------------------------------------------------------------------------

class RealDistExponential;

class RealDistExponentialReporter : public DistribReporter
{
    public:
                    RealDistExponentialReporter (const RealDistExponential&);
        virtual    ~RealDistExponentialReporter ();
    protected:
        virtual ostream&    writeParamA (ostream&) const;
};

// ---------------------------------------------------------------------------

class RealDistErlang;

class RealDistErlangReporter : public DistribReporter
{
    public:
                    RealDistErlangReporter (const RealDistErlang&);
        virtual    ~RealDistErlangReporter ();
    protected:
        virtual ostream&    writeParamA (ostream&) const;
        virtual ostream&    writeParamB (ostream&) const;
```

```
};

// --------------------------------------------------------------------------

class RealDistNormal;

class RealDistNormalReporter : public DistribReporter
{
    public:
                    RealDistNormalReporter (const RealDistNormal&);
        virtual    ~RealDistNormalReporter ();
    protected:
        virtual ostream&    writeParamA (ostream&) const;
        virtual ostream&    writeParamB (ostream&) const;
};

// --------------------------------------------------------------------------

class RealDistEmpirical;

class RealDistEmpiricalReporter : public DistribReporter
{
    public:
                    RealDistEmpiricalReporter (const RealDistEmpirical&);
        virtual    ~RealDistEmpiricalReporter ();

        virtual ostream&    WriteAsLine (ostream&, int reportWidth) const;
        virtual ostream&    WriteAsBlock(ostream&) const;
    protected:
};

// --------------------------------------------------------------------------

#endif  // DISTRIBREPORTER_H
```

repdist.cc

```
// --------------------------------------------------------------------------
//
// Datei
//          repdist.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// --------------------------------------------------------------------------

#include "repdist.h"

#include "distribu.h"
#include "booldist.h"
#include "intdist.h"
#include "realdist.h"
#include <iostream.h>
#include <iomanip.h>

const cTypeWidth        = 11;
const cParamWidth       = 12;
const cParamPrecision   =  4;
const cSeedWidth        =  9;

// --------------------------------------------------------------------------

unsigned DistribReporter::groupID = NewGroupID();

// --------------------------------------------------------------------------

DistribReporter::DistribReporter (const Distribution& d)
    :   Reporter(d)
```

```
    {}

// ------------------------------------------------------------------------------

DistribReporter::~DistribReporter ()
    {}

// ------------------------------------------------------------------------------

unsigned DistribReporter::GetGroupID() const
    {   return groupID; }

// ------------------------------------------------------------------------------

String DistribReporter::GetTitle() const
    {   return "Distributions"; }

// ------------------------------------------------------------------------------

ostream& DistribReporter::writeType (ostream &o) const
    {
        Distribution& dist = (Distribution&)reportable;
        return
            o   << ' ' << setw(cTypeWidth) << setiosflags(ios::left)
                << dist.GetType().Left (cTypeWidth)
                << resetiosflags(ios::left);
    }

// ------------------------------------------------------------------------------

ostream& DistribReporter::writeParamA (ostream &o) const
    {
        return o
            << ' ' << setw(cParamWidth) << "";
    }

// ------------------------------------------------------------------------------

ostream& DistribReporter::writeParamB (ostream &o) const
    {
        return o
            << ' ' << setw(cParamWidth) << "";
    }

// ------------------------------------------------------------------------------

ostream& DistribReporter::writeParam (ostream &o, double p) const
    {
        long flg = o.flags(ios::showpoint | ios::fixed | ios::right);

        o << ' ' << setw(cParamWidth) << setprecision(cParamPrecision) << p;
        o.flags(flg);
        return o;
    }

// ------------------------------------------------------------------------------

ostream& DistribReporter::writeParam (ostream &o, int p) const
    {
        o   << ' ' << setw(cParamWidth - cParamPrecision -1) << p
            << setw(cParamPrecision +1) << "";
        return o;
    }

// ------------------------------------------------------------------------------

ostream& DistribReporter::writeSeed (ostream &o) const
    {
        Distribution& dist = (Distribution&)reportable;
        return o << ' ' << setw(cSeedWidth) << dist.Seed();
    }

// ------------------------------------------------------------------------------

int DistribReporter::LineEnd() const
    {
        return Reporter::LineEnd() + NameWidth() + 2 * cParamWidth
                + cSeedWidth + 4;
    }

// ------------------------------------------------------------------------------

ostream& DistribReporter::WriteHeader (ostream& os) const
    {
```

```
        return Reporter::WriteHeader (os)
            << " " << setw (cTypeWidth) << setiosflags (ios::left)
            << String("Type").Left (cTypeWidth)
            << resetiosflags (ios::left)
            << " " << setw (2 * cParamWidth + 1)
            << String("P a r a m e t e r s").Left (2 * cParamWidth + 1)
            << " " << setw (cSeedWidth)
            << String("Seed").Left (cSeedWidth);
    }

// ---------------------------------------------------------------------------

ostream& DistribReporter::UnderscoreHeader (ostream& os) const
    {
        char    ofill = os.fill();

        return Reporter::UnderscoreHeader (os)
            << setfill('-') << setw (4  + cTypeWidth + 2 * cParamWidth
                                        + cSeedWidth) << ""
            << setfill(ofill);
    }

// ---------------------------------------------------------------------------

ostream& DistribReporter::WriteAsLine (ostream& os, int reportWidth) const
    {
        Distribution&   dist = (Distribution&)reportable;

        Reporter::WriteAsLine (os, reportWidth);
        writeType (os);
        writeParamA (os);
        writeParamB (os);
        writeSeed (os);
        return os;
    }

// ---------------------------------------------------------------------------

ostream& DistribReporter::WriteAsBlock (ostream& os) const
    {
        Distribution&   dist = (Distribution&)reportable;

        Reporter::WriteAsBlock (os) << endl
            << "Type: ";        writeType (os) << endl
            << "1. Parameter: ";writeParamA (os) << endl
            << "2. Parameter: ";writeParamB (os) << endl
            << "Seed: ";        writeSeed (os) << endl;
        return os;
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

BoolDistConstReporter::BoolDistConstReporter (const BoolDistConst& d)
    :   DistribReporter(d)
    {}

// ---------------------------------------------------------------------------

BoolDistConstReporter::~BoolDistConstReporter ()
    {}

// ---------------------------------------------------------------------------

ostream& BoolDistConstReporter::writeParamA (ostream& os) const
    {
        BoolDistConst& dist = (BoolDistConst&)reportable;
        return os
            << ' ' << setw (cParamWidth - cParamPrecision -1)
            << (dist.GetValue() ? "true" : "false")
            << setw (cParamPrecision +1) << "";
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

BoolDistBernoulliReporter::BoolDistBernoulliReporter
                               (const BoolDistBernoulli& d)
    :   DistribReporter(d)
    {}

// ---------------------------------------------------------------------------
```

```
BoolDistBernoulliReporter::~BoolDistBernoulliReporter ()
    {}

// -------------------------------------------------------------------------

ostream& BoolDistBernoulliReporter::writeParamA (ostream& os) const
    {
        BoolDistBernoulli& dist = (BoolDistBernoulli&)reportable;
        return writeParam (os, dist.GetProbability());
    }

// -------------------------------------------------------------------------
// -------------------------------------------------------------------------
// -------------------------------------------------------------------------

IntDistConstReporter::IntDistConstReporter (const IntDistConst& d)
    :   DistribReporter(d)
    {}

// -------------------------------------------------------------------------

IntDistConstReporter::~IntDistConstReporter ()
    {}

// -------------------------------------------------------------------------

ostream& IntDistConstReporter::writeParamA (ostream& os) const
    {
        IntDistConst& dist = (IntDistConst&)reportable;
        return writeParam (os, dist.GetValue());
    }

// -------------------------------------------------------------------------
// -------------------------------------------------------------------------

IntDistUniformReporter::IntDistUniformReporter (const IntDistUniform& d)
    :   DistribReporter(d)
    {}

// -------------------------------------------------------------------------

IntDistUniformReporter::~IntDistUniformReporter ()
    {}

// -------------------------------------------------------------------------

ostream& IntDistUniformReporter::writeParamA (ostream& os) const
    {
        IntDistUniform& dist = (IntDistUniform&)reportable;
        return writeParam (os, dist.GetLow());
    }

// -------------------------------------------------------------------------

ostream& IntDistUniformReporter::writeParamB (ostream& os) const
    {
        IntDistUniform& dist = (IntDistUniform&)reportable;
        return writeParam (os, dist.GetHigh());
    }

// -------------------------------------------------------------------------
// -------------------------------------------------------------------------

IntDistPoissonReporter::IntDistPoissonReporter (const IntDistPoisson& d)
    :   DistribReporter(d)
    {}

// -------------------------------------------------------------------------

IntDistPoissonReporter::~IntDistPoissonReporter ()
    {}

// -------------------------------------------------------------------------

ostream& IntDistPoissonReporter::writeParamA (ostream& os) const
    {
        IntDistPoisson& dist = (IntDistPoisson&)reportable;
        return writeParam (os, dist.GetMean());
    }

// -------------------------------------------------------------------------
// -------------------------------------------------------------------------

IntDistEmpiricalReporter::IntDistEmpiricalReporter (const IntDistEmpirical& d)
```

```
    :   DistribReporter(d)
    {}

// ------------------------------------------------------------------------------

IntDistEmpiricalReporter::~IntDistEmpiricalReporter ()
    {}

// ------------------------------------------------------------------------------

ostream& IntDistEmpiricalReporter::WriteAsLine (ostream& os,
                                                int      reportWidth) const
    {
        IntDistEmpirical&   dist = (IntDistEmpirical&)reportable;
        unsigned    n       = dist.CountEntries() -1;
        const       begin   = Reporter::LineEnd() + cTypeWidth + 1;

        Reporter::WriteAsLine (os, reportWidth);
        writeType (os);
        if (n < 0)
        {
            writeParamA (os);
            writeParamB (os);
            writeSeed (os);
        } else
        {
            writeParam (os, dist.GetValue (0));
            writeParam (os, dist.GetCumulativeFrequency (0));
            writeSeed (os);
            for (int i = 1; i <= n; i++)
            {
                os << endl << setw (begin) << "";
                writeParam (os, dist.GetValue (i));
                writeParam (os, dist.GetCumulativeFrequency (i));
            }
        }
        return os;
    }

// ------------------------------------------------------------------------------

ostream& IntDistEmpiricalReporter::WriteAsBlock (ostream& os) const
    {
        IntDistEmpirical&   dist = (IntDistEmpirical&)reportable;
        char    ofill = os.fill();

        Reporter::WriteAsBlock (os) << endl
            << "Type: ";        writeType (os) << endl
            << "Seed: ";        writeSeed (os) << endl
            << setw(cParamWidth +1) << String("Value").Left(cParamWidth)
            << setw(cParamWidth +1) << String("Cum. Freq.").Left(cParamWidth)
            << endl << setfill('-') << setw (2 * cParamWidth +1) << ""
            << setfill(ofill) << endl;
        for (int i = 0; i < dist.CountEntries(); i++)
        {
            writeParam (os, dist.GetValue (i));
            writeParam (os, dist.GetCumulativeFrequency (i)) << endl;
        }
        return os;
    }

// ------------------------------------------------------------------------------
// ------------------------------------------------------------------------------
// ------------------------------------------------------------------------------

RealDistConstReporter::RealDistConstReporter (const RealDistConst& d)
    :   DistribReporter(d)
    {}

// ------------------------------------------------------------------------------

RealDistConstReporter::~RealDistConstReporter ()
    {}

// ------------------------------------------------------------------------------

ostream& RealDistConstReporter::writeParamA (ostream& os) const
    {
        RealDistConst& dist = (RealDistConst&)reportable;
        return writeParam (os, dist.GetValue());
    }

// ------------------------------------------------------------------------------
// ------------------------------------------------------------------------------
```

```
RealDistUniformReporter::RealDistUniformReporter (const RealDistUniform& d)
    :   DistribReporter(d)
    {}

// --------------------------------------------------------------------------

RealDistUniformReporter::~RealDistUniformReporter ()
    {}

// --------------------------------------------------------------------------

ostream& RealDistUniformReporter::writeParamA (ostream& os) const
    {
        RealDistUniform& dist = (RealDistUniform&)reportable;
        return writeParam (os, dist.GetLow());
    }

// --------------------------------------------------------------------------

ostream& RealDistUniformReporter::writeParamB (ostream& os) const
    {
        RealDistUniform& dist = (RealDistUniform&)reportable;
        return writeParam (os, dist.GetHigh());
    }

// --------------------------------------------------------------------------
// --------------------------------------------------------------------------

RealDistNormalReporter::RealDistNormalReporter (const RealDistNormal& d)
    :   DistribReporter(d)
    {}

// --------------------------------------------------------------------------

RealDistNormalReporter::~RealDistNormalReporter ()
    {}

// --------------------------------------------------------------------------

ostream& RealDistNormalReporter::writeParamA (ostream& os) const
    {
        RealDistNormal& dist = (RealDistNormal&)reportable;
        return writeParam (os, dist.GetMean());
    }

// --------------------------------------------------------------------------

ostream& RealDistNormalReporter::writeParamB (ostream& os) const
    {
        RealDistNormal& dist = (RealDistNormal&)reportable;
        return writeParam (os, dist.GetStddev());
    }

// --------------------------------------------------------------------------
// --------------------------------------------------------------------------

RealDistErlangReporter::RealDistErlangReporter (const RealDistErlang& d)
    :   DistribReporter(d)
    {}

// --------------------------------------------------------------------------

RealDistErlangReporter::~RealDistErlangReporter ()
    {}

// --------------------------------------------------------------------------

ostream& RealDistErlangReporter::writeParamA (ostream& os) const
    {
        RealDistErlang& dist = (RealDistErlang&)reportable;
        return writeParam (os, dist.GetMean());
    }

// --------------------------------------------------------------------------

ostream& RealDistErlangReporter::writeParamB (ostream& os) const
    {
        RealDistErlang& dist = (RealDistErlang&)reportable;
        return writeParam (os, int(dist.GetK()));
    }

// --------------------------------------------------------------------------
// --------------------------------------------------------------------------
```

```
RealDistExponentialReporter::RealDistExponentialReporter
                                 (const RealDistExponential& d)
    :   DistribReporter(d)
    {}

// ---------------------------------------------------------------------------

RealDistExponentialReporter::~RealDistExponentialReporter ()
    {}

// ---------------------------------------------------------------------------

ostream& RealDistExponentialReporter::writeParamA (ostream& os) const
    {
        RealDistExponential& dist = (RealDistExponential&)reportable;
        return writeParam (os, dist.GetMean());
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

RealDistEmpiricalReporter::RealDistEmpiricalReporter
                               (const RealDistEmpirical& d)
    :   DistribReporter(d)
    {}

// ---------------------------------------------------------------------------

RealDistEmpiricalReporter::~RealDistEmpiricalReporter ()
    {}

// ---------------------------------------------------------------------------

ostream& RealDistEmpiricalReporter::WriteAsLine (ostream& os,
                                                 int      reportWidth) const
    {
        RealDistEmpirical&  dist = (RealDistEmpirical&)reportable;
        unsigned    n       = dist.CountEntries() -1;
        const       begin   = Reporter::LineEnd() + cTypeWidth + 1;

        Reporter::WriteAsLine (os, reportWidth);
        writeType (os);
        if (n < 0)
        {
            writeParamA (os);
            writeParamB (os);
            writeSeed (os);
        } else
        {
            writeParam (os, dist.GetValue (0));
            writeParam (os, dist.GetCumulativeFrequency (0));
            writeSeed (os);
            for (int i = 1; i <= n; i++)
            {
                os << endl << setw (begin) << "";
                writeParam (os, dist.GetValue (i));
                writeParam (os, dist.GetCumulativeFrequency (i));
            }
        }
        return os;
    }

// ---------------------------------------------------------------------------

ostream& RealDistEmpiricalReporter::WriteAsBlock (ostream& os) const
    {
        RealDistEmpirical&  dist = (RealDistEmpirical&)reportable;
        char    ofill = os.fill();

        Reporter::WriteAsBlock (os) << endl
            << "Type: ";        writeType (os) << endl
            << "Seed: ";        writeSeed (os) << endl
            << setw(cParamWidth +1) << String("Value").Left(cParamWidth)
            << setw(cParamWidth +1) << String("Cum. Freq.").Left(cParamWidth)
            << endl << setfill('-') << setw (2 * cParamWidth +1) << ""
            << setfill(ofill) << endl;
        for (int i = 0; i < dist.CountEntries(); i++)
        {
            writeParam (os, dist.GetValue (i));
            writeParam (os, dist.GetCumulativeFrequency (i)) << endl;
        }
        return os;
    }
```

```
// --------------------------------------------------------------------------
```

repmodel.h

```
// --------------------------------------------------------------------------
//
// Datei
//          repmodel.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// --------------------------------------------------------------------------

#ifndef MODELREPORTER_H
#define MODELREPORTER_H

// --------------------------------------------------------------------------

#include "reporter.h"   // Basisklasse

#include "str.h"

// --------------------------------------------------------------------------

class ReportableList;
class ReporterList;

// --------------------------------------------------------------------------

class ModelReporter : public Reporter
{
    public:
                    ModelReporter (const Model&, ReportableList&);
        virtual    ~ModelReporter ();

        virtual unsigned    GetGroupID() const;
        virtual String      GetTitle() const;
        virtual bool        HasTitle() const;   // Vorgabe = true
        virtual bool        HasHeader() const;  // Vorgabe = true

        virtual ostream&    WriteHeader (ostream&) const;
        virtual ostream&    UnderscoreHeader (ostream&) const;
        virtual ostream&    WriteAsLine (ostream&, int reportWidth) const;
        virtual ostream&    WriteAsBlock(ostream&) const;
    protected:

    private:
                ReporterList&   reporterList;
        static  unsigned        groupID;
};

// --------------------------------------------------------------------------

#endif  // MODELREPORTER_H
```

repmodel.cc

```
// --------------------------------------------------------------------------
//
// Datei
//          repmodel.cc
```

```
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#include "repmodel.h"

#include "boolean.h"
#include "experimm.h"
#include "messagem.h"
#include "msgtypes.h"
#include "model.h"
#include "reportal.h"   // ReportableList
#include "ring.h"
#include "str.h"
#include <assert.h>

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

class ReporterList : public Ring<Reporter>
{
    public:
                ReporterList (ReportableList& rl);
        virtual ~ReporterList ();
};

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

ReporterList::ReporterList (ReportableList& reportableList)
    //  zu jedem Reportable aus der 'reportableList' wird ein Reporter
    //  in die ReporterList einsortiert (nach Gruppe, chronologisch)
    //  reportableList ist ein Stack => von hinten abarbeiten
    //  dann stimmt die chronologische Reihenfolge wieder
    {
        Reportable* reportable = reportableList.Last();

        for (int i = reportableList.Size(); i > 0; i--)
        {   // fuer jedes reportbare Objekt:
            if (reportable->ShowInReport())
            {
                Reporter*   toInsert = reportable->NewReporter();
                if (toInsert)
                {
                    bool        InsertionPointfound = false;
                    int         j                   = Size();
                    Reporter*   current             = First ();

                    while (j && !InsertionPointfound)
                    {   // suche Reporter, vor dem eingefuegt werden soll
                        if (*toInsert < *current)
                        {
                            // als letzten seiner Gruppe eintragen
                            Insert (toInsert);  // vor current
                            InsertionPointfound = true;
                        }
                        else
                        {
                            current = Next();
                            j--;
                        }
                    }
                    if (!InsertionPointfound)
                        Append (toInsert);  // anhaengen
                }
            }
            reportable = reportableList.Prev();
        }
    }

// ---------------------------------------------------------------------------

ReporterList::~ReporterList ()
```

```
    {
        for (int i = Size(); i > 0; i--)
            delete Dequeue();   // Reporter loeschen
    }

// ------------------------------------------------------------------------------

class ModelStartEndMessage : public ReportMessage
{
    public:
        ModelStartEndMessage (const String& modelName, bool start)
            :   ReportMessage (String(start ? "Model: " : "End of Model: ")
                                + modelName,
                                Message::descriptionAsBox)
            {};
};

// ------------------------------------------------------------------------------

unsigned ModelReporter::groupID = (unsigned)-1; // MAX => als letzte auflisten

// ------------------------------------------------------------------------------

ModelReporter::ModelReporter (const Model& m, ReportableList& rl)
    :   Reporter(m),
        reporterList(*new ReporterList(rl))
    {}

// ------------------------------------------------------------------------------

ModelReporter::~ModelReporter ()
    {   delete &reporterList; }

// ------------------------------------------------------------------------------

unsigned ModelReporter::GetGroupID () const
    {   return groupID; }

// ------------------------------------------------------------------------------

String ModelReporter::GetTitle() const
    {   return "Models"; }

// ------------------------------------------------------------------------------

bool ModelReporter::HasTitle () const
    {   return false; }

// ------------------------------------------------------------------------------

bool ModelReporter::HasHeader () const
    {   return true; }

// ------------------------------------------------------------------------------

ostream& ModelReporter::WriteHeader (ostream& os) const
    {   return os; }

// ------------------------------------------------------------------------------

ostream& ModelReporter::UnderscoreHeader (ostream& os) const
    {   return os; }

// ------------------------------------------------------------------------------

ostream& ModelReporter::WriteAsLine (ostream& os, int reportWidth) const
    {
        Model&  model = (Model&)reportable;
        MessageManager& mm = ExperimentManager::Instance().
                                GetMessageManager(model);
        mm.Note (ModelStartEndMessage (GetName(), true));// Anfang des Modells

        os << GetName() << " reset at: " << reportable.ResetAt()
           << endl << endl;

        String description = model.Description();
        if (description.Length() > 0)
            os << description << endl;

        Reporter* reporter = reporterList.First();

        for (int i = reporterList.Size(); i > 0; i--)
        {
            assert (reporter);
```

```
            mm.TakeReporter(*reporter);
            reporter = reporterList.Next();
        }

        mm.Note (ModelStartEndMessage (GetName(), false));// Ende des Modells
        return os;
    }

// --------------------------------------------------------------------------

ostream& ModelReporter::WriteAsBlock (ostream& os) const
    {
        Model& model = (Model&)reportable;

        return os << model.Description();
    }

// --------------------------------------------------------------------------
```

reportab.h

```
// --------------------------------------------------------------------------
//
// Datei
//         reportab.h
//
// Diplomarbeit
//
//         DESMO-C
//         Implementierung eines Simulators fuer
//         zeitdiskrete Simulation in C++
//
// Autor
//         Thomas Schniewind
//
// Datum
//         8.3.1998
//
// --------------------------------------------------------------------------

#ifndef REPORTABLE_H
#define REPORTABLE_H

// --------------------------------------------------------------------------

#include "modelcom.h"   // Basisklasse

#include "boolean.h"
#include "simtime.h"
#include "str.h"

// --------------------------------------------------------------------------

class Reporter;

// --------------------------------------------------------------------------

class Reportable : public ModelComponent
    /*  Reportable dient als Oberklasse fuer alle Objekte ueber
        die im Ergebnis-Report Informationen dargestellt werden
        sollen. Ein solches Objekt muss auf Verlangen einen Reporter
        erzeugen koennen (Methode NewReporter), der die Informationen
        fuer den Ergebnis-Report entsprechend aufbereitet.
    */
{
    public:
                    Reportable (const Reportable&); // Kopierkonstruktor
                    Reportable (Model& owner, const String& name,
                                bool showInReport   = true,
                                bool showInTrace    = true);
        virtual    ~Reportable ();

                bool            ShowInReport() const;   // abfragen
                void            ShowInReport(bool);     // setzen

                unsigned long   Observations() const;

        virtual void            Reset ();
```

```
                SimTime         ResetAt () const;

        virtual Reporter*       NewReporter () const = 0;

                String          ClassName () const;

    protected:
                void            IncObservations (unsigned long inc = 1);
    private:
                Reportable&     operator= (const Reportable&);
                                    // Zuweisung nicht implementiert

                bool            showInReport;
                SimTime         resetAt;
                unsigned long   observations;
};

// ---------------------------------------------------------------------------

#endif  // REPORTABLE_H
```

reportab.cc

```
// ---------------------------------------------------------------------------
//
// Datei
//          reportab.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#include "reportab.h"

#include "experime.h"
#include "experimm.h"
#include "model.h"

#include <iomanip.h>

// ---------------------------------------------------------------------------

static const char* className = "Reportable";

// ---------------------------------------------------------------------------

Reportable::Reportable (Model& owner, const String& name,
                        bool SHOWInReport, bool showInTrace)
    :   ModelComponent  (owner, name, showInTrace),
        showInReport    (SHOWInReport),
        resetAt         (0.0),
        observations    (0)
    {
        if (this != &owner) // Hauptmodell: Reset dort
            Reset();
        // Anmelden:
        ExperimentManager::Instance().Register (*this);
    }

// ---------------------------------------------------------------------------

Reportable::Reportable (const Reportable& r)
    :   ModelComponent  (r),
        showInReport    (r.showInReport),
        resetAt         (r.resetAt),
        observations    (r.observations)
    {
        // Anmelden
```

```
        ExperimentManager::Instance().Register (*this);
    }

// ---------------------------------------------------------------------------

Reportable::~Reportable ()
    {
        // Abmelden
        ExperimentManager::Instance().DeRegister (*this);
    }

// ---------------------------------------------------------------------------

bool Reportable::ShowInReport() const
    {
        return showInReport;
    }

// ---------------------------------------------------------------------------

void Reportable::ShowInReport(bool show)
    {
        showInReport = show;
    }

// ---------------------------------------------------------------------------

void Reportable::Reset()
    {
        const char* where = "Reportable::Reset";

        if (!valid (className, where))
            return;

        resetAt = CurrentTime();
        observations = 0;
    }

// ---------------------------------------------------------------------------

SimTime Reportable::ResetAt() const
    {
        return resetAt;
    }

// ---------------------------------------------------------------------------

unsigned long Reportable::Observations() const
    {
        return observations;
    }

// ---------------------------------------------------------------------------

String Reportable::ClassName () const
    {
        return className;
    }

// ---------------------------------------------------------------------------

void Reportable::IncObservations (unsigned long add)
    {
        observations += add;
    }

// ---------------------------------------------------------------------------
```

reportal.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          reportal.h
//
// Diplomarbeit
//
//          DESMO-C
```

```
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// --------------------------------------------------------------------------

#ifndef REPORTABLELIST_H
#define REPORTABLELIST_H

// --------------------------------------------------------------------------

#include "ring.h"

// --------------------------------------------------------------------------

class ReportableList    : public Ring<Reportable>   {};

// --------------------------------------------------------------------------

#endif  // REPORTABLELIST_H
```

reporter.h

```
// --------------------------------------------------------------------------
//
// Datei
//          reporter.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// --------------------------------------------------------------------------

#ifndef REPORTER_H
#define REPORTER_H

// --------------------------------------------------------------------------

#include "boolean.h"
#include "str.h"

// --------------------------------------------------------------------------

class Model;
class Reportable;

// --------------------------------------------------------------------------

class Reporter
{
    public:
                    Reporter (const Reportable&);
        virtual    ~Reporter ();


                Model&      GetModel() const;
        virtual unsigned    GetGroupID() const = 0;
                String      GetName() const;    // Name des Reportables

        virtual String      GetTitle() const = 0;
        virtual bool        HasTitle() const;   // Vorgabe = true

                ostream&    WriteTitle (ostream&, int width) const;
                                // width <= Title.Length ==> zentriert,
```

```
                                // sonst linksb.

                ostream&    UnderscoreTitle (ostream&, int width) const;
                                // width <= Title.Length ==> zentriert,
                                // sonst linksb.

        virtual bool        HasHeader() const;  // Vorgabe = true
        virtual ostream&    WriteHeader (ostream&) const;
        virtual ostream&    UnderscoreHeader (ostream&) const;

        virtual ostream&    WriteAsLine (ostream&, int reportWidth) const;
        virtual ostream&    WriteAsBlock(ostream&) const;

                bool        operator<  (const Reporter&);
                bool        operator<= (const Reporter&);
                bool        operator>  (const Reporter&);
                bool        operator>= (const Reporter&);
    protected:
        static  unsigned    NewGroupID();
                int         NameWidth() const;
                int         TimeWidth() const;
                int         TimePrecision () const;
        virtual int         LineEnd() const;

        const   Reportable& reportable;
    private:
        static  unsigned    nextGroupID;
};

// ----------------------------------------------------------------------------

#endif  // REPORTER_H
```

reporter.cc

```
// ----------------------------------------------------------------------------
//
// Datei
//          reporter.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ----------------------------------------------------------------------------

#include "reporter.h"

#include "experime.h"
#include "model.h"
#include "reportab.h"
#include <iostream.h>
#include <iomanip.h>

// ----------------------------------------------------------------------------

unsigned Reporter::nextGroupID = 1;

// ----------------------------------------------------------------------------

Reporter::Reporter (const Reportable& r)
    : reportable(r)
    {}

// ----------------------------------------------------------------------------

Reporter::~Reporter ()
    {}

// ----------------------------------------------------------------------------
```

```
Model& Reporter::GetModel() const
    { return reportable.GetModel(); }

// ---------------------------------------------------------------------------

String Reporter::GetName() const
    { return reportable.Name(); }

// ---------------------------------------------------------------------------

bool Reporter::HasTitle () const
    {   return true; }

// ---------------------------------------------------------------------------

ostream& Reporter::WriteTitle (ostream& os, int width) const
    {
        String  title   = GetTitle();
        if (title.Length() > 0)
        {
            char    ofill   = os.fill();
            int     length  = title.Length();

            if (width > length)
                os  << setw((width - length) / 2) << ""
                    << title << endl;
            else
                os  << title << endl;
        }
        return os;
    };

// ---------------------------------------------------------------------------

ostream& Reporter::UnderscoreTitle (ostream& os, int width) const
    {
        int     length  = GetTitle().Length();
        if (length > 0)
        {
            char    ofill   = os.fill();

            if (width > length)
                os  << setw((width - length) / 2) << ""
                    << setfill('-') << setw(length) << ""
                    << setfill(ofill) << endl;
            else
                os  << setfill('-') << setw(length) << ""
                    << setfill(ofill) << endl;
        }
        return os;
    };

// ---------------------------------------------------------------------------

bool Reporter::HasHeader () const
    {   return true; }

// ---------------------------------------------------------------------------

const obsWidth = 6;

// ---------------------------------------------------------------------------

ostream& Reporter::WriteHeader (ostream& os) const
    {
        return os
            << setw (NameWidth()) << setiosflags (ios::left)
            << String("Title").Left (NameWidth())
            << resetiosflags (ios::left)
            << " " << setw (TimeWidth())
            << String("(Re)set").Left (TimeWidth()) << " "
            << setw (obsWidth)
            << String("Obs").Left (obsWidth);
    }

// ---------------------------------------------------------------------------

ostream& Reporter::UnderscoreHeader (ostream& os) const
    {
        char    ofill = os.fill();

        return os
            << setfill('-')
```

```
            << setw (NameWidth() + 2 + TimeWidth() + obsWidth) << ""
            << setfill(ofill);
    }

// ---------------------------------------------------------------------------

ostream& Reporter::WriteAsLine (ostream& os, int reportWidth) const
    {
        return os
            << setw (NameWidth()) << setiosflags (ios::left)
            << reportable.Name().Left (NameWidth())
            << resetiosflags (ios::left)
            << setw (TimeWidth() +1)
            << reportable.ResetAt().AsString (TimeWidth(), TimePrecision()-1)
            << setw (obsWidth +1) << reportable.Observations();
    }

// ---------------------------------------------------------------------------

ostream& Reporter::WriteAsBlock (ostream& os) const
    {
        return os
            << "Title: " << reportable.Name() << endl
            << "(Re)set at: " << reportable.ResetAt() << endl
            << "Observations: " << reportable.Observations();
    }

// ---------------------------------------------------------------------------

bool Reporter::operator< (const Reporter& r)
    {
        unsigned    id  = GetGroupID();
        unsigned    rid = r.GetGroupID();
        return id < rid;
    }

// ---------------------------------------------------------------------------

bool Reporter::operator<= (const Reporter& r)
    {
        unsigned    id  = GetGroupID();
        unsigned    rid = r.GetGroupID();
        return id <= rid;
    }

// ---------------------------------------------------------------------------

bool Reporter::operator> (const Reporter& r)
    { return ! (*this <= r); }

// ---------------------------------------------------------------------------

bool Reporter::operator>= (const Reporter& r)
    { return ! (*this < r); }

// ---------------------------------------------------------------------------

unsigned Reporter::NewGroupID()
    { return nextGroupID++; }

// ---------------------------------------------------------------------------

int Reporter::NameWidth() const
    { return reportable.GetExperimentOpts().NameWidth(); }

// ---------------------------------------------------------------------------

int Reporter::TimeWidth() const
    { return reportable.GetExperimentOpts().TimeWidth(); }

// ---------------------------------------------------------------------------

int Reporter::TimePrecision() const
    { return reportable.GetExperimentOpts().TimePrecision(); }

// ---------------------------------------------------------------------------

int Reporter::LineEnd() const
    { return NameWidth() + TimeWidth() + obsWidth + 2; }

// ---------------------------------------------------------------------------
```

repqueue.h

```
// -------------------------------------------------------------------------
//
// Datei
//          repqueue.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// -------------------------------------------------------------------------

#ifndef QUEUEREPORTER_H
#define QUEUEREPORTER_H

// -------------------------------------------------------------------------

#include "reporter.h"   // Basisklasse

#include "str.h"

// -------------------------------------------------------------------------

class QueueBased;

// -------------------------------------------------------------------------

class QueueReporter : public Reporter
{
    public:
                    QueueReporter (const QueueBased&);
        virtual    ~QueueReporter ();

        virtual unsigned    GetGroupID() const;
        virtual String      GetTitle () const;

        virtual ostream&    WriteHeader (ostream&) const;
        virtual ostream&    UnderscoreHeader (ostream&) const;
        virtual ostream&    WriteAsLine (ostream&, int reportWidth) const;
        virtual ostream&    WriteAsBlock(ostream&) const;

    private:
        static  unsigned    groupID;
};

// -------------------------------------------------------------------------

#endif  // QUEUEREPORTER_H
```

repqueue.cc

```
// -------------------------------------------------------------------------
//
// Datei
//          repqueue.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
```

```
//
// ---------------------------------------------------------------------------

#include "repqueue.h"

#include "qbased.h"
#include <iostream.h>
#include <iomanip.h>

// ---------------------------------------------------------------------------

const cQmaxWidth        = 7;
const cQnowWidth        = 7;
const cQavgWidth        = 10;
const cQavgPrecision    = 3;
const cZeroWaitsWidth   = 6;
const cAvgWaitWidth     = 10;
const cAvgWaitPrecision = 3;

// ---------------------------------------------------------------------------

unsigned QueueReporter::groupID = NewGroupID();

// ---------------------------------------------------------------------------

QueueReporter::QueueReporter (const QueueBased& q)
    :   Reporter(q)
    {}

// ---------------------------------------------------------------------------

QueueReporter::~QueueReporter ()
    {}

// ---------------------------------------------------------------------------

unsigned QueueReporter::GetGroupID() const
    {   return groupID; }

// ---------------------------------------------------------------------------

String QueueReporter::GetTitle() const
    {   return "Queues"; }

// ---------------------------------------------------------------------------

ostream& QueueReporter::WriteHeader (ostream& os) const
    {
        return Reporter::WriteHeader (os)
            << " " << setw (cQmaxWidth)
            << String("Qmax").Left (cQmaxWidth)
            << " " << setw (cQnowWidth)
            << String("Qnow").Left (cQnowWidth)
            << " " << setw (cQavgWidth)
            << String("Qavg.").Left (cQavgWidth)
            << " " << setw (cZeroWaitsWidth)
            << String("Zeros").Left (cZeroWaitsWidth)
            << " " << setw (cAvgWaitWidth)
            << String("avg.Wait").Left (cAvgWaitWidth);
    }

// ---------------------------------------------------------------------------

ostream& QueueReporter::UnderscoreHeader (ostream& os) const
    {
        char    ofill = os.fill();

        return Reporter::UnderscoreHeader (os)
            << setfill('-')
            << setw (5  + cQmaxWidth + cQnowWidth + cQavgWidth
                        + cZeroWaitsWidth + cAvgWaitWidth)
            << ""
            << setfill(ofill);
    }

// ---------------------------------------------------------------------------

ostream& QueueReporter::WriteAsLine (ostream& os, int reportWidth) const
    {
        long    oflgs = os.flags(ios::showpoint | ios::fixed | ios::right);
        QueueBased& queue = (QueueBased&)reportable;

        return Reporter::WriteAsLine (os, reportWidth)
            << setw (cQmaxWidth +1) << queue.MaxLength()
```

```
            << setw (cQnowWidth +1) << queue.Length()
            << setw (cQavgWidth +1) << setprecision (cQavgPrecision)
            << setiosflags(ios::showpoint | ios::fixed) << queue.AvgLength()
            << setw (cZeroWaitsWidth +1) << queue.ZeroWaits()
            << setw (cAvgWaitWidth +1) << queue.AvgWaitTime()
            << setiosflags (oflgs);
    }

// ---------------------------------------------------------------------------

ostream& QueueReporter::WriteAsBlock (ostream& os) const
    {
        QueueBased& queue = (QueueBased&)reportable;

        return Reporter::WriteAsBlock (os) << endl
            << "Qmax: " << queue.MaxLength() << endl
            << "Qnow: " << queue.Length() << endl
            << "Qavg.: " << queue.AvgLength() << endl
            << "Zeros: " << queue.ZeroWaits() << endl
            << "avg.Wait: " << queue.AvgWaitTime();
    }

// ---------------------------------------------------------------------------
```

represbi.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          represbi.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#ifndef RESBINREPORTER_H
#define RESBINREPORTER_H

// ---------------------------------------------------------------------------

#include "reporter.h"   // Basisklasse

#include "str.h"

// ---------------------------------------------------------------------------

class QueueBased;

class ResBinReporter : public Reporter
{
    public:
                    ResBinReporter (const QueueBased&);

        virtual ostream&    WriteHeader (ostream&) const;
        virtual ostream&    UnderscoreHeader (ostream&) const;
        virtual ostream&    WriteAsLine (ostream&, int reportWidth) const;
        virtual ostream&    WriteAsBlock(ostream&) const;

        virtual String      GetHeaderUsers      () const;
        virtual unsigned    GetUsers            () const = 0;
        virtual String      GetHeaderInit       () const = 0;
        virtual unsigned    GetInit             () const = 0;
        virtual String      GetHeaderExtreme    () const = 0;
        virtual unsigned    GetExtreme          () const = 0;
        virtual String      GetHeaderNow        () const;
        virtual unsigned    GetNow              () const = 0;
        virtual String      GetHeaderMean       () const = 0;
        virtual double      GetMean             () const = 0;
```

```
    protected:
        virtual int         LineEnd() const;

    private:
                String      BlockTitle (const String&) const;

        static  unsigned    groupID;
};

// --------------------------------------------------------------------------

class Bin;

class BinReporter : public ResBinReporter
{
    public:
                    BinReporter (const Bin&);

        virtual unsigned    GetGroupID() const;
        virtual String      GetTitle () const;

        virtual unsigned    GetUsers            () const;
        virtual String      GetHeaderInit       () const;
        virtual unsigned    GetInit             () const;
        virtual String      GetHeaderExtreme    () const;
        virtual unsigned    GetExtreme          () const;
        virtual unsigned    GetNow              () const;
        virtual String      GetHeaderMean       () const;
        virtual double      GetMean             () const;

    private:
        static  unsigned    groupID;
};

// --------------------------------------------------------------------------

class Res;

class ResReporter : public ResBinReporter
{
    public:
                    ResReporter (const Res&);

        virtual unsigned    GetGroupID() const;
        virtual String      GetTitle () const;

        virtual unsigned    GetUsers            () const;
        virtual String      GetHeaderInit       () const;
        virtual unsigned    GetInit             () const;
        virtual String      GetHeaderExtreme    () const;
        virtual unsigned    GetExtreme          () const;
        virtual unsigned    GetNow              () const;
        virtual String      GetHeaderMean       () const;
        virtual double      GetMean             () const;

    private:
        static  unsigned    groupID;
};

// --------------------------------------------------------------------------

#endif  // RESBINREPORTER_H
```

represbi.cc

```
// --------------------------------------------------------------------------
//
// Datei
//          represbi.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
```

```
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ----------------------------------------------------------------------------

#include "represbi.h"

#include "bin.h"
#include "res.h"

#include <iostream.h>
#include <iomanip.h>

// ----------------------------------------------------------------------------

const cUsrWidth     =  7;
const cIinitWidth   =  7;
const cExtWidth     =  7;
const cNowWidth     =  7;
const cMeanWidth    = 11;
const cMeanPrec     =  3;
const cAvgWaitWidth = 11;
const cAvgWaitPrec  =  3;
const cQMaxLWidth   =  6;

// ----------------------------------------------------------------------------
// ----------------------------------------------------------------------------

ResBinReporter::ResBinReporter (const QueueBased& qb)
    :   Reporter (qb)
    {}

// ----------------------------------------------------------------------------

ostream& ResBinReporter::WriteHeader (ostream& os) const
    {
        return os
            << setw (NameWidth())   << setiosflags (ios::left)
                                    << String("Title").Left (NameWidth())
            << resetiosflags (ios::left)
            << ' ' << setw (TimeWidth())
                                    << String("(Re)set").Left (TimeWidth())

            << setw (cUsrWidth)     << GetHeaderUsers().Left (cUsrWidth)
            << setw (cIinitWidth)   << GetHeaderInit().Left (cIinitWidth)
            << setw (cExtWidth)     << GetHeaderExtreme().Left (cExtWidth)
            << setw (cNowWidth)     << GetHeaderNow().Left (cNowWidth)
            << setw (cMeanWidth) << setprecision (cMeanPrec)
                                    << GetHeaderMean().Left (cMeanWidth)

            << setw (cAvgWaitWidth) << String("avg.Wait").Left (cAvgWaitWidth)
            << setw (cQMaxLWidth)   << String("QMaxL").Left (cQMaxLWidth);
    }

// ----------------------------------------------------------------------------

ostream& ResBinReporter::UnderscoreHeader (ostream& os) const
    {
        char    ofill = os.fill();

        return os
            << setfill('-')
            << setw (NameWidth() + 1 + TimeWidth()
                        + cUsrWidth
                        + cIinitWidth
                        + cExtWidth
                        + cNowWidth
                        + cMeanWidth
                        + cAvgWaitWidth
                        + cQMaxLWidth)
            << "" << setfill(ofill);
    }

// ----------------------------------------------------------------------------

ostream& ResBinReporter::WriteAsLine (ostream& os, int reportWidth) const
    {
        QueueBased& qb   = (QueueBased&)reportable;
        long        oflgs = os.flags(ios::showpoint | ios::fixed | ios::right);

        return os
            << setw (NameWidth()) << setiosflags (ios::left)
```

```
            << qb.Name().Left (NameWidth())
            << resetiosflags (ios::left)
            << setw (TimeWidth() +1)
            << qb.ResetAt().AsString (TimeWidth(), TimePrecision()-1)

            << setw (cUsrWidth) << GetUsers()
            << setw (cIinitWidth) << GetInit()
            << setw (cExtWidth) << GetExtreme()
            << setw (cNowWidth) << GetNow()
            << setw (cMeanWidth) << setprecision(cMeanPrec) << GetMean()

            << setw (cAvgWaitWidth) << setprecision(cAvgWaitPrec)
            << qb.AvgWaitTime()
            << setw (cQMaxLWidth) << qb.MaxLength()
            << setiosflags (oflgs);
    }

// ------------------------------------------------------------------------------

ostream& ResBinReporter::WriteAsBlock (ostream& os) const
    {
        QueueBased& qb = (QueueBased&)reportable;
        long oflgs = os.flags(ios::showpoint | ios::fixed | ios::right);
        const width = 12;

        return os
            << BlockTitle ("Title")            << qb.Name()
                                               << endl
            << BlockTitle ("(Re)set at")       << setw(10) << qb.ResetAt()
                                               << endl
            << BlockTitle (GetHeaderUsers())   << setw(10) << GetUsers()
                                               << endl
            << BlockTitle (GetHeaderInit())    << setw(10) << GetInit()
                                               << endl
            << BlockTitle (GetHeaderExtreme()) << setw(10) << GetExtreme()
                                               << endl
            << BlockTitle (GetHeaderNow())     << setw(10) << GetNow()
                                               << endl
            << BlockTitle (GetHeaderMean())    << setw(10) << GetMean()
                                               << endl
            << BlockTitle ("AvgWait")          << setw(10) << qb.AvgWaitTime()
                                               << endl
            << BlockTitle ("Zeros")            << setw(10) << qb.ZeroWaits()
                                               << endl
            << BlockTitle ("Max. QLength")     << setw(10) << qb.MaxLength()
                                               << endl
            << BlockTitle ("Avg. QLength")     << setw(10) << qb.AvgLength()
            << setiosflags (oflgs);
    }

// ------------------------------------------------------------------------------

String ResBinReporter::BlockTitle (const String& string) const
    {
        strstream ss;
        ss.flags(ios::left);
        ss << setw(12) << string << ": " << ends;
        return ss;
    }

// ------------------------------------------------------------------------------

int ResBinReporter::LineEnd() const
    {
        return  NameWidth()     + TimeWidth() + 2
                + cUsrWidth     + cIinitWidth
                + cExtWidth     + cNowWidth
                + cMeanWidth    + cAvgWaitWidth
                + cQMaxLWidth;
    }

// ------------------------------------------------------------------------------

String ResBinReporter::GetHeaderUsers() const
    {
        return  "Users";
    }

// ------------------------------------------------------------------------------

String ResBinReporter::GetHeaderNow() const
    {
        return  "Now";
    }
```

```
// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

unsigned BinReporter::groupID = NewGroupID();

// ---------------------------------------------------------------------------

BinReporter::BinReporter (const Bin& b)
    :   ResBinReporter(b)
    {}

// ---------------------------------------------------------------------------

unsigned BinReporter::GetGroupID() const
    {   return groupID; }

// ---------------------------------------------------------------------------

String BinReporter::GetTitle() const
    {   return "Bins"; }

// ---------------------------------------------------------------------------

String BinReporter::GetHeaderInit() const
    {   return "Init"; }

// ---------------------------------------------------------------------------

String BinReporter::GetHeaderExtreme() const
    {   return "Max"; }

// ---------------------------------------------------------------------------

String BinReporter::GetHeaderMean() const
    {   return "Average"; }

// ---------------------------------------------------------------------------

unsigned BinReporter::GetUsers() const
    {
        return ((Bin&)reportable).Producers();
    }

// ---------------------------------------------------------------------------

unsigned BinReporter::GetInit() const
    {
        return ((Bin&)reportable).Initial();
    }

// ---------------------------------------------------------------------------

unsigned BinReporter::GetExtreme() const
    {
        return ((Bin&)reportable).Maximum();
    }

// ---------------------------------------------------------------------------

unsigned BinReporter::GetNow() const
    {
        return ((Bin&)reportable).Avail();
    }

// ---------------------------------------------------------------------------

double BinReporter::GetMean() const
    {
        return ((Bin&)reportable).AvgAvail();
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

unsigned ResReporter::groupID = NewGroupID();

// ---------------------------------------------------------------------------

ResReporter::ResReporter (const Res& r)
    :   ResBinReporter(r)
    {}

// ---------------------------------------------------------------------------
```

```
unsigned ResReporter::GetGroupID() const
    {   return groupID; }

// ----------------------------------------------------------------------------

String ResReporter::GetTitle() const
    {   return "Resources"; }

// ----------------------------------------------------------------------------

String ResReporter::GetHeaderInit() const
    {   return "Limit"; }

// ----------------------------------------------------------------------------

String ResReporter::GetHeaderExtreme() const
    {   return "Min"; }

// ----------------------------------------------------------------------------

String ResReporter::GetHeaderMean() const
    {   return "Usage [%]"; }

// ----------------------------------------------------------------------------

unsigned ResReporter::GetUsers() const
    {
        return ((Res&)reportable).Users();
    }

// ----------------------------------------------------------------------------

unsigned ResReporter::GetInit() const
    {
        return ((Res&)reportable).Limit();
    }

// ----------------------------------------------------------------------------

unsigned ResReporter::GetExtreme() const
    {
        return ((Res&)reportable).Minimum();
    }

// ----------------------------------------------------------------------------

unsigned ResReporter::GetNow() const
    {
        return ((Res&)reportable).Avail();
    }

// ----------------------------------------------------------------------------

double ResReporter::GetMean() const
    {
        return ((Res&)reportable).AvgUsage() * 100.0;
    }

// ----------------------------------------------------------------------------
```

repstat.h

```
// ----------------------------------------------------------------------------
//
// Datei
//          repstat.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
```

```
//          8.3.1998
//
// ---------------------------------------------------------------------------

#ifndef STATISTICREPORTER_H
#define STATISTICREPORTER_H

// ---------------------------------------------------------------------------

#include "reporter.h"   // Basisklasse

#include "str.h"

// ---------------------------------------------------------------------------

class Count;

class CountReporter : public Reporter
{
    public:
                    CountReporter (const Count& c);

        virtual unsigned    GetGroupID() const;
        virtual String      GetTitle() const;

    private:
        static  unsigned    groupID;
};

// ---------------------------------------------------------------------------

class Regression;

class RegressionReporter : public Reporter
{
    public:
                    RegressionReporter (const Regression&);
        virtual    ~RegressionReporter ();

        virtual unsigned    GetGroupID() const;
        virtual String      GetTitle() const;

        virtual bool        HasHeader() const;
        virtual ostream&    WriteHeader (ostream&) const;
        virtual ostream&    UnderscoreHeader (ostream&) const;
        virtual ostream&    WriteAsLine (ostream&, int reportWidth) const;
        virtual ostream&    WriteAsBlock(ostream&) const;

    protected:
        virtual int         LineEnd() const;

    private:
        static  unsigned    groupID;
};

// ---------------------------------------------------------------------------

class ValueStatistics;

class ValueStatisticsReporter : public Reporter
{
    public:
        virtual    ~ValueStatisticsReporter ();

        virtual ostream&    WriteHeader (ostream&) const;
        virtual ostream&    UnderscoreHeader (ostream&) const;
        virtual ostream&    WriteAsLine (ostream&, int reportWidth) const;
        virtual ostream&    WriteAsBlock(ostream&) const;
    protected:
                    ValueStatisticsReporter (const ValueStatistics&);

        virtual int         LineEnd() const;

    private:
};

// ---------------------------------------------------------------------------

class Accumulate;

class AccumulateReporter : public ValueStatisticsReporter
{
    public:
                    AccumulateReporter (const Accumulate&);
```

```
        virtual    ~AccumulateReporter ();

        virtual unsigned    GetGroupID() const;
        virtual String      GetTitle() const;
    private:
        static  unsigned    groupID;
};

// ----------------------------------------------------------------------------

class Tally;

class TallyReporter : public ValueStatisticsReporter
{
    public:
                    TallyReporter (const Tally&);
        virtual    ~TallyReporter ();

        virtual unsigned    GetGroupID() const;
        virtual String      GetTitle() const;
    private:
        static  unsigned    groupID;
};

// ----------------------------------------------------------------------------

class Histogram;

class HistogramReporter : public TallyReporter
{
    public:
                    HistogramReporter (const Histogram&);
        virtual    ~HistogramReporter ();

        virtual unsigned    GetGroupID() const;
        virtual String      GetTitle() const;

        virtual bool        HasHeader() const;
        virtual ostream&    WriteHeader (ostream&) const;
        virtual ostream&    UnderscoreHeader (ostream&) const;
        virtual ostream&    WriteAsLine (ostream&, int reportWidth) const;
        virtual ostream&    WriteAsBlock(ostream&) const;

    protected:
        virtual int         LineEnd() const;

    private:
                ostream&    HistogramHeader (ostream&   os,
                                             int        reportWidth) const;
                ostream&    HistogramBody   (ostream&   os,
                                             int        reportWidth) const;

        static  unsigned    groupID;
};

// ----------------------------------------------------------------------------

#endif  // STATISTICREPORTER_H
```

repstat.cc

```
// ----------------------------------------------------------------------------
//
// Datei
//          repstat.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
```

```
// ---------------------------------------------------------------------------

#include "repstat.h"

#include "accumula.h"
#include "boolean.h"
#include "count.h"
#include "histogra.h"
#include "regress.h"
#include "tally.h"

#include <iostream.h>
#include <iomanip.h>

// ---------------------------------------------------------------------------

const cWidth        = 10;
const cPrecision    =  3;

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------
// CountReporter:

unsigned CountReporter::groupID = NewGroupID();

// ---------------------------------------------------------------------------

CountReporter::CountReporter (const Count& c)
    :   Reporter(c)
    {}

// ---------------------------------------------------------------------------

unsigned CountReporter::GetGroupID() const
    {
        return groupID;
    }

// ---------------------------------------------------------------------------

String CountReporter::GetTitle() const
    {
        return "Counts";
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------
// RegressionReporter:

unsigned RegressionReporter::groupID = NewGroupID();

// ---------------------------------------------------------------------------

RegressionReporter::RegressionReporter (const Regression& a)
    :   Reporter(a)
    {}

// ---------------------------------------------------------------------------

RegressionReporter::~RegressionReporter ()
    {}

// ---------------------------------------------------------------------------

unsigned RegressionReporter::GetGroupID() const
    {
        return groupID;
    }

// ---------------------------------------------------------------------------

String RegressionReporter::GetTitle() const
    {
        return "Regression";
    }

// ---------------------------------------------------------------------------

bool RegressionReporter::HasHeader () const
    {
        return false;
    }

// ---------------------------------------------------------------------------
```

```
ostream& RegressionReporter::WriteHeader (ostream& os) const
    {
        return os;
    }

// --------------------------------------------------------------------------

ostream& RegressionReporter::UnderscoreHeader (ostream& os) const
    {
        return os;
    }

// --------------------------------------------------------------------------

ostream& RegressionReporter::WriteAsLine (ostream& os, int reportWidth) const
    {
        Regression& reg = (Regression&)reportable;
        long  oflgs = os.flags(ios::showpoint | ios::fixed | ios::right);
        char  ofill = os.fill();
        const String cHeader =    "Regression of " + reg.QuotedName() +
                                  " upon " + reg.QuotedName2();
        const   String  cInsuf =  "***  Insufficient Data  ***";
        const   String  cDegen =  "***  Degenerate Data  ***";
        const   String  cConst =  " = constant = ";
        const   int     cCharW =  18;
        const   int     cBarW  =  (cHeader.Length() > cCharW + 2 + cWidth) ?
                                        cHeader.Length()
                                    :   cCharW + 2 + cWidth;

        os  << cHeader << endl
            << setfill ('-') << setw (cBarW) << ""
            << setfill (ofill)
            << endl << "(Re)set           : "
            << setw (cWidth) << setprecision (cPrecision)
            << reg.ResetAt().Time()
            << endl << "Obs               : "
            << setw (cWidth - cPrecision - 1) << reg.Observations();
        if (reg.Observations() > 0)
            os  << endl << "xBar              : "
                << setw (cWidth) << setprecision (cPrecision) << reg.xMean()
                << endl << "yBar              : "
                << setw (cWidth) << setprecision (cPrecision) << reg.yMean();
        if (reg.Observations() <= 5)
            os  << endl << cInsuf;
        else
        {
            if (reg.xConstant() || reg.yConstant())
            {
                os  << endl << cDegen;
                if (reg.xConstant())
                    os  << endl << 'x' << cConst
                        << setiosflags(ios::showpoint | ios::fixed)
                        << setw(cWidth) << setprecision(cPrecision)
                        << reg.xMean()
                        << resetiosflags(ios::showpoint | ios::fixed);
                if (reg.yConstant())
                    os  << endl << 'y' << cConst
                        << setiosflags(ios::showpoint | ios::fixed)
                        << setw(cWidth) << setprecision(cPrecision)
                        << reg.yMean()
                        << resetiosflags(ios::showpoint | ios::fixed);
            }
            else
                os  << setiosflags(ios::showpoint | ios::fixed)
                    << endl << "Res.St.Dev        : "
                    << setw(cWidth) << setprecision(cPrecision)
                    << reg.ResStdDev()
                    << endl << "Reg.Coeff         : "
                    << setw(cWidth) << setprecision(cPrecision)
                    << reg.RegCoeff()
                    << endl << "Intercept         : "
                    << setw(cWidth) << setprecision(cPrecision)
                    << reg.Intercept()
                    << endl << "St.Dev.Reg.Coeff  : "
                    << setw(cWidth) << setprecision(cPrecision)
                    << reg.StdDevRegCoeff()
                    << endl << "Corr.Coeff        : "
                    << setw(cWidth) << setprecision(cPrecision)
                    << reg.CorrCoeff()
                    << resetiosflags(ios::showpoint | ios::fixed);
        }
        return os << setiosflags (oflgs);
    }
```

```
// -------------------------------------------------------------------------

ostream& RegressionReporter::WriteAsBlock (ostream& os) const
    {
        return WriteAsLine (os, 0);
    }

// -------------------------------------------------------------------------

int RegressionReporter::LineEnd () const
    {
        return 0;
    }

// -------------------------------------------------------------------------
// -------------------------------------------------------------------------
// ValueStatisticsReporter:

ValueStatisticsReporter::ValueStatisticsReporter (const ValueStatistics& v)
    :   Reporter(v)
    {}

// -------------------------------------------------------------------------

ValueStatisticsReporter::~ValueStatisticsReporter ()
    {}

// -------------------------------------------------------------------------

int ValueStatisticsReporter::LineEnd() const
    {
        return Reporter::LineEnd() + 4 * cWidth + 4;
    }

// -------------------------------------------------------------------------

ostream& ValueStatisticsReporter::WriteHeader (ostream& os) const
    {
        return Reporter::WriteHeader (os)
            << setiosflags (ios::right)
            << ' ' << setw (cWidth) << String("Mean").Left (cWidth)
            << ' ' << setw (cWidth) << String("Std.Dev").Left (cWidth)
            << ' ' << setw (cWidth) << String("Min").Left (cWidth)
            << ' ' << setw (cWidth) << String("Max").Left (cWidth);
    }

// -------------------------------------------------------------------------

ostream& ValueStatisticsReporter::UnderscoreHeader (ostream& os) const
    {
        char    ofill = os.fill();

        return Reporter::UnderscoreHeader (os)
            << setfill('-') << setw (4 * cWidth + 4) << ""
            << setfill(ofill);
    }

// -------------------------------------------------------------------------

ostream& ValueStatisticsReporter::WriteAsLine (ostream& os,
                                               int      reportWidth) const
    {
        ValueStatistics&    vsr = (ValueStatistics&)reportable;
        long    oflgs = os.flags(ios::showpoint | ios::fixed | ios::right);

        return Reporter::WriteAsLine (os, reportWidth)
            << setw (cWidth +1) << setprecision (cPrecision) << vsr.Mean()
            << setw (cWidth +1) << setprecision (cPrecision) << vsr.StdDev()
            << setw (cWidth +1) << setprecision (cPrecision) << vsr.Minimum()
            << setw (cWidth +1) << setprecision (cPrecision) << vsr.Maximum()
            << setiosflags (oflgs);
    }

// -------------------------------------------------------------------------

ostream& ValueStatisticsReporter::WriteAsBlock (ostream& os) const
    {
        ValueStatistics&    vsr = (ValueStatistics&)reportable;
        long    oflgs = os.flags(ios::showpoint | ios::fixed | ios::right);

        return Reporter::WriteAsBlock (os)
            << endl << "Mean: "
            << setw (cWidth +1) << setprecision (cPrecision) << vsr.Mean()
```

```
            << endl << "Std.Dev: "
            << setw (cWidth +1) << setprecision (cPrecision) << vsr.StdDev()
            << endl << "Min: "
            << setw (cWidth +1) << setprecision (cPrecision) << vsr.Minimum()
            << endl << "Max: "
            << setw (cWidth +1) << setprecision (cPrecision) << vsr.Maximum()
            << setiosflags (oflgs);
    }

// ----------------------------------------------------------------------------
// ----------------------------------------------------------------------------
// AccumulateReporter:

unsigned AccumulateReporter::groupID = NewGroupID();

// ----------------------------------------------------------------------------

AccumulateReporter::AccumulateReporter (const Accumulate& a)
    :   ValueStatisticsReporter(a)
    {}

// ----------------------------------------------------------------------------

AccumulateReporter::~AccumulateReporter ()
    {}

// ----------------------------------------------------------------------------

unsigned AccumulateReporter::GetGroupID() const
    {
        return groupID;
    }

// ----------------------------------------------------------------------------

String AccumulateReporter::GetTitle() const
    {
        return "Accumulates";
    }

// ----------------------------------------------------------------------------
// ----------------------------------------------------------------------------
// TallyReporter:

unsigned TallyReporter::groupID = NewGroupID();

// ----------------------------------------------------------------------------

TallyReporter::TallyReporter (const Tally& t)
    :   ValueStatisticsReporter(t)
    {}

// ----------------------------------------------------------------------------

TallyReporter::~TallyReporter ()
    {}

// ----------------------------------------------------------------------------

unsigned TallyReporter::GetGroupID() const
    {
        return groupID;
    }

// ----------------------------------------------------------------------------

String TallyReporter::GetTitle() const
    {
        return "Tallies";
    }

// ----------------------------------------------------------------------------
// ----------------------------------------------------------------------------
// HistogramReporter:

unsigned HistogramReporter::groupID = NewGroupID();

// ----------------------------------------------------------------------------

HistogramReporter::HistogramReporter (const Histogram& h)
    :   TallyReporter(h)
    {}

// ----------------------------------------------------------------------------
```

```
HistogramReporter::~HistogramReporter ()
    {}

// ---------------------------------------------------------------------------

unsigned HistogramReporter::GetGroupID() const
    {
        return groupID;
    }

// ---------------------------------------------------------------------------

String HistogramReporter::GetTitle() const
    {
        return "Histograms";
    }

// ---------------------------------------------------------------------------

bool HistogramReporter::HasHeader () const
    {
        return false;
    }

// ---------------------------------------------------------------------------

ostream& HistogramReporter::WriteHeader (ostream& os) const
    {
        return os;
    }

// ---------------------------------------------------------------------------

ostream& HistogramReporter::UnderscoreHeader (ostream& os) const
    {
        return os;
    }

// ---------------------------------------------------------------------------

ostream& HistogramReporter::WriteAsLine (ostream& os, int reportWidth) const
    {
        Histogram&      hist  = (Histogram&)reportable;

        TallyReporter::WriteHeader (os) << endl;
        TallyReporter::UnderscoreHeader (os) << endl;
        TallyReporter::WriteAsLine (os, reportWidth) << endl << endl;

        if (reportWidth < 60)
            reportWidth = 60;

        HistogramHeader (os, reportWidth) << endl;

        if (hist.Observations() == 0)
            return os << "***  No entries recorded  ***" << endl;

        return HistogramBody (os, reportWidth) << endl;
    }

// ---------------------------------------------------------------------------

const cCellWidth    =  4;
const cLowerWidth   = 11;
const cLowerPrec    =  3;
const cObsWidth     =  7;
const cPercWidth    =  9;
const cPercPrec     =  3;
const cLeftCol      = cCellWidth + cLowerWidth + cObsWidth
                        + 2 * cPercWidth + 2;

// ---------------------------------------------------------------------------

ostream& HistogramReporter::HistogramHeader (ostream& os,
                                             int      reportWidth) const
    {
        long oflgs = os.flags(ios::showpoint | ios::fixed | ios::right);
        char            ofill = os.fill();
        const String    cCell = "Cell";
        const String    cLow  = "Lower Lim";
        const String    cObs  = "N";
        const String    cPerc = "%";
        const String    cCum  = "Cum%";
        return os
```

```
            << setw (cCellWidth) << cCell.Left (cCellWidth)
            << setw (cLowerWidth) << cLow.Left (cLowerWidth)
            << setw (cObsWidth) << cObs.Left (cObsWidth)
            << setw (cPercWidth) << cPerc.Left (cPercWidth)
            << setw (cPercWidth) << cCum.Left (cPercWidth)
            << endl << setfill('-')
            << setw (reportWidth) << ""
            << setfill(ofill) << setiosflags (oflgs);
    }

// ---------------------------------------------------------------------------

ostream& HistogramReporter::HistogramBody ( ostream&    os,
                                            int         reportWidth) const
    {
        Histogram&      hist  = (Histogram&)reportable;
        long oflgs = os.flags(ios::showpoint | ios::fixed | ios::right);
        char            ofill = os.fill();
        const String    cInf  = "-Infintiy";
        double          low   = hist.Lower(0),
                        sum   = 0.0,
                        scale = double(reportWidth - cLeftCol)
                                / double (hist.ObservationsInCell (
                                            hist.MostFrequentedCell()));
        u_long          total = 0;
        bool            restIsEmpty = false;

        for (unsigned cell = 0; cell < hist.Cells() + 2; cell++)
        {
            u_long  obsInCell   = hist.ObservationsInCell (cell);
            u_long  obs         = hist.Observations ();
            double  frequency   = double (obsInCell) / double (obs) * 100.0;

            // Cell
            os  << setw (cCellWidth) << cell;

            // Lower Limit
            if (cell == 0)
                os  << setw (cLowerWidth) << cInf.Left (cLowerWidth);
            else
                os  << setw (cLowerWidth) << setprecision (cLowerPrec)
                    << hist.Lower (cell);

            // Observations
            os  << setw (cObsWidth) << obsInCell;

            // %
            os  << setw (cPercWidth) << setprecision (cPercPrec) << frequency;

            // Cum%
            sum += frequency;
            os  << setw (cPercWidth) << setprecision (cPercPrec)
                << sum << " |";

            // Data
            if (restIsEmpty)
            {
                os  << "   +++ Rest of Table empty +++" << endl;
                break;
            }
            os  << setw (unsigned (scale * obsInCell + 0.5)) << setfill ('*')
                << "" << setfill (ofill) << endl;

            total += obsInCell;
            if (total == obs && cell < hist.Cells())
                restIsEmpty = true;
        } // for

        return os
            << setw (reportWidth) << setfill ('-') << ""
            << setfill (ofill) << setiosflags (oflgs);
    }

// ---------------------------------------------------------------------------

ostream& HistogramReporter::WriteAsBlock (ostream& os) const
    {
        return WriteAsLine (os, cLeftCol + 10);
    }

// ---------------------------------------------------------------------------

int HistogramReporter::LineEnd () const
    {
```

```
        return 0;
    }

// ----------------------------------------------------------------------------
```

repwaitq.h

```
// ----------------------------------------------------------------------------
//
// Datei
//          repwaitq.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ----------------------------------------------------------------------------

#ifndef WAITQUEUEREPORTER_H
#define WAITQUEUEREPORTER_H

// ----------------------------------------------------------------------------

#include "repqueue.h"   // Basisklasse

#include "str.h"

// ----------------------------------------------------------------------------

class WaitQueue;

// ----------------------------------------------------------------------------

class WaitQueueReporter : public QueueReporter
{
                    WaitQueueReporter (const WaitQueueReporter&);
    public:
                    WaitQueueReporter (const WaitQueue&);
        virtual     ~WaitQueueReporter ();

        virtual unsigned    GetGroupID() const;
        virtual String      GetTitle() const;

        virtual ostream&    WriteAsLine (ostream&, int reportWidth) const;
        virtual ostream&    WriteAsBlock(ostream&) const;

    private:
        static  unsigned    groupID;

                Reporter*   slaveQueueReporter;
};

// ----------------------------------------------------------------------------

#endif  // WAITQUEUEREPORTER_H
```

repwaitq.cc

```
// ----------------------------------------------------------------------------
//
// Datei
//          repwaitq.cc
//
// Diplomarbeit
```

```
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#include "repwaitq.h"

#include "waitq.h"

// ---------------------------------------------------------------------------

unsigned WaitQueueReporter::groupID = NewGroupID();

// ---------------------------------------------------------------------------

WaitQueueReporter::WaitQueueReporter (const WaitQueue& wq)
    :   QueueReporter       (wq),
        slaveQueueReporter  (wq.waitingSlaves.NewReporter())
    {}

// ---------------------------------------------------------------------------

WaitQueueReporter::~WaitQueueReporter ()
    {
        delete slaveQueueReporter;
    }

// ---------------------------------------------------------------------------

unsigned WaitQueueReporter::GetGroupID() const
    {
        return groupID;
    }

// ---------------------------------------------------------------------------

String WaitQueueReporter::GetTitle() const
    {
        return "Wait-Queues";
    }

// ---------------------------------------------------------------------------

ostream& WaitQueueReporter::WriteAsLine (ostream& os, int reportWidth) const
    {
        QueueReporter::WriteAsLine (os, reportWidth);
        os << endl;
        slaveQueueReporter->WriteAsLine (os, reportWidth);
        return os << endl;
    }

// ---------------------------------------------------------------------------

ostream& WaitQueueReporter::WriteAsBlock(ostream& os) const
    {
        QueueReporter::WriteAsBlock (os);
        os << endl;
        slaveQueueReporter->WriteAsBlock (os);
        return os;
    }

// ---------------------------------------------------------------------------
```

res.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          res.h
//
```

```
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// -----------------------------------------------------------------------

#ifndef RES_H
#define RES_H

// -----------------------------------------------------------------------

#include "qbased.h" // Basisklasse

#include "boolean.h"
#include "simtime.h"
#include "str.h"

// -----------------------------------------------------------------------

class Res : public QueueBased
{
            Res&    operator= (const Res&); // nicht implementiert
    public:
                    Res (Model& owner, const String& name = "",
                         unsigned long capacity     = 1,
                         bool          showInReport = true,
                         bool          showInTrace  = true);
                    Res (const Res&);
        virtual    ~Res ();

                void            Acquire (unsigned long n);
                void            Release (unsigned long n);

                void            ChangeLimit (unsigned long n);
                                    // kann nur geaendert werden, wenn
                                    // noch nicht benutzt

                unsigned long   Limit () const;
                unsigned long   Avail () const;
                unsigned long   Minimum () const;
                unsigned long   Users () const;
                double          AvgUsage () const;

                bool            ReleasedAll (Process& p, unsigned long& n);
                                    // hat p alle Einheiten zurueckgegeben?
        virtual void            Reset();
        virtual Reporter*       NewReporter () const;

                String          ClassName () const;

    private:
                bool        checkProcess (Process&      p,
                                          const char*   where) const;
                void        updateStatistics (long n);
                void        activateNext (const char* where) const;

        // von ResourceDB benutzte Methoden:
        friend class ResourceDB;
                void        ReleaseAllWithWarning   (Process&,
                                                     unsigned long wants,
                                                     unsigned long holds);
                void        ConsumerNotFound        (Process&,
                                                     unsigned long);
                void        ConsumerReleaseTooMuch  (Process&,
                                                     unsigned long has,
                                                     unsigned long wants);

        QueueImpl&          qimpl;
        unsigned long       limit,
                            avail,
                            minimum,
                            users;
        double              wSumAvail;
        SimTime             lastReturn;
        ResourceDB&         resourceDB;
};
```

```
// --------------------------------------------------------------------------

#endif  // RES_H
```

res.cc

```
// --------------------------------------------------------------------------
//
// Datei
//          res.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// --------------------------------------------------------------------------

#include "res.h"

#include "experimm.h"
#include "model.h"
#include "pblocker.h"
#include "process.h"
#include "qimpl.h"
#include "represbi.h"   // Reporter
#include "resdb.h"
#include "msgresbi.h"

#include <assert.h>

// --------------------------------------------------------------------------

static const char* className = "Res";

// --------------------------------------------------------------------------

Res::Res (  Model& owner,   const String& name,
            unsigned long   n,
            bool            showInReport,
            bool            showInTrace)
    :   QueueBased  (owner, name, showInReport, showInTrace),
        qimpl       (*new QueueImpl (*this)),
        limit       (n),
        avail       (n),
        minimum     (n),
        users       (0),
        wSumAvail   (0.0),
        lastReturn  (CurrentTime()),
        resourceDB  (ExperimentManager::Instance().
                         GetResourceDB((ModelComponent&)*this))
    {
        const char* where = "Res::Res()";
        if (n < 1)
        {
            Warning ("illegal limit for Res", where, "limit is set to 1");
            avail = minimum = 1;
        }
    }

// --------------------------------------------------------------------------

Res::Res (const Res& r)
    :   QueueBased  (r),
        qimpl       (*new QueueImpl (*this)),
        limit       (r.limit),
        avail       (r.limit),  // alle verfuegbar
        minimum     (r.minimum),
        users       (r.users),
        wSumAvail   (r.wSumAvail),
```

```
        lastReturn  (r.lastReturn),
        resourceDB  (r.resourceDB)
    {}

// -----------------------------------------------------------------------------

Res::~Res ()
    {
        delete &qimpl;
    }

// -----------------------------------------------------------------------------

void Res::Reset()
    {
        QueueBased::Reset();
        if (!Valid ()) return;

        wSumAvail   =
        minimum     =
        users       = 0;
        lastReturn  = CurrentTime();
    }

// -----------------------------------------------------------------------------

Reporter* Res::NewReporter () const
    {
        return new ResReporter (*this);
    }

// -----------------------------------------------------------------------------

String Res::ClassName () const
    {
        return className;
    }

// -----------------------------------------------------------------------------

bool Res::checkProcess (Process& p, const char* where) const
    {
        if (!p.Valid ())
        {
            Warning ("invalid object", where);
            return false;
        }
        if (p.IsNullProcess())
        {
            Warning ("only processes may acquire or release resources",
                     where);
            return false;
        }
        if (!IsExperimentCompatible (p))
        {
            Warning ("attemp to mix components of different experiments",
                     where, "ignored");
            return false;
        }
        if (!IsModelCompatible (p))
        {
            Warning ("incompatible process", where);
            return false;
        }
        return true;
    }

// -----------------------------------------------------------------------------

void Res::updateStatistics (long n)
    {
        SimTime now = CurrentTime();

        wSumAvail += double(avail) * (now - lastReturn).Time();
        lastReturn = now;
        avail += n;
        if (avail < minimum)
            minimum = avail;
    }

// -----------------------------------------------------------------------------

void Res::activateNext (const char* where) const
    {
```

```
        if (Length() > 0)
        {
            Process& next = (Process&)qimpl.First (where);
            if (!checkProcess (next, where))
                return;
            if (next.IsScheduled())
            {   // anders als in DESMO!
                next.SkipTraceNote ();
                next.Cancel();
            }


            bool wasBlocked = next.Blocked();
            if (wasBlocked)
                ProcessBlocker::UnBlock (next);
                    // um Aktivierung zu erlauben

            next.SkipTraceNote ();
            next.ActivateAfter (Current());

            if (wasBlocked) ProcessBlocker::SetBlocked (next);
        }
    }

// ---------------------------------------------------------------------------

void Res::Acquire (unsigned long n)
    {
        const char*     where = "Res::Acquire";

        if (!valid (className, where))
            return;

        bool            activateSuccessor = false;
        Process&        process = CurrentProcess();
        if (!checkProcess (process, where))
            return;

        if (n <= 0)
        {
            if (TraceIsOn())
                SendMessage (TrcResSeize (*this, n));
            return;
        }

        // Anforderung von n Einheiten anmelden,
        // und Gesamtzahl (hold + want) zurueck bekommen:
        unsigned long total =
        resourceDB.AddWaiter (this, &process, n);

        if (total > limit)
        {
            if (total == n)
                Warning (process.QuotedName() +
                         " tries to acquire more than limit",
                         where,
                         "call is ignored, " + process.QuotedName()
                         + " is not blocked");
            else
                Warning (process.QuotedName() + " tries to acquire "
                         + String(n) + " units from Resource "
                         + QuotedName() + " while it holds " +
                         String(total - n) + " units, which would "
                         "be more than the limit of "
                         + String(limit) + " units.",
                         where, "ignored, process is not blocked");
            // Anforderung widerrufen:
            resourceDB.DelWaiter (this, &process, n);
            return;
        }

        // DEADLOCK CHECK (Dynamic A + Static)
        if (resourceDB.DynamicCheckA (this, &process, process.Err()))
            Warning (   "Deadlock detected while "
                        + process.QuotedName() + " acquires "
                        + String(n) + " units of " + QuotedName(),
                        where);

        qimpl.Insert (process, where);

        if (n > avail ||                    // nicht genug vorhanden
            process != qimpl.First (where)) // es wartet noch ein anderer
        {
            if (TraceIsOn())
```

```
                SendMessage (TrcResBinAwait (*this, n));

            // DEADLOCK CHECK (Dynamic B)
            if (resourceDB.DynamicCheckB (this, &process, Err()))
                Warning (   "Deadlock detected while "
                            + process.QuotedName() + " acquires "
                            + String(n) + " units of " + QuotedName(),
                            where);

            for (;;)
            {
                ProcessBlocker::Block (process);

                if (n <= avail && process == qimpl.First (where))
                    // jetzt genuegend frei und kein anderer vorher dran
                    break;       // for (;;) verlassen
            }

            activateSuccessor = true;
        }

        // allocate in resource map
        resourceDB.AddConsumer (this, &process, n); // loescht Waiter

        if (TraceIsOn())
            SendMessage (TrcResSeize (*this, n));

        qimpl.Remove (process, where);
        ProcessBlocker::UnBlock (process);

        // nachdem current aus Queue entfernt:
        if (activateSuccessor)
            activateNext (where);

        updateStatistics (-n);  // -n fuer 'Acquire (n)'
    }

// ----------------------------------------------------------------------------

void Res::Release (unsigned long n)
    {
        const char* where = "Res::Release";

        if (!valid (className, where))
            return;

        Process& process = CurrentProcess();
        if (!checkProcess (process, where))
            return;

        unsigned long   used = resourceDB.AskConsumer (this, &process);
        if (n > used)
        {
            Warning (   process.QuotedName() + " wants to release more to "
                        + QuotedName() + " than it holds" ,
                        where,
                        "all [" + String(used) + "] will be released");
            n = used;
        }

        if (TraceIsOn())
            SendMessage (TrcResRelease (*this, n));

        updateStatistics (n);
        users++;

        resourceDB.DelConsumer (this, &process, n);

        activateNext (where);
    }

// ----------------------------------------------------------------------------

void Res::ChangeLimit (unsigned long n)
    {
        const char* where = "Res::ChangeLimit";

        if (!valid (className, where))
            return;

        if (avail != limit || Observations() > 0)
        {
            Warning (   "Attempt to change limit of already used Res " +
                            QuotedName(),
```

```
                        where,
                        "will be ignored",
                        "after 'Reset', you can change the limit again, if "
                        "no units are in use");
            return;
        }

        if (n == 0)
        {
            Warning (   "Attempt to set the limit of " + QuotedName() +
                        " to zero",
                        where,
                        "will be ignored");
            return;
        }

        limit   =
        avail   =
        minimum = n;
    }

// ----------------------------------------------------------------------------

unsigned long Res::Limit () const
    {
        return limit;
    }

// ----------------------------------------------------------------------------

unsigned long Res::Avail () const
    {
        return avail;
    }

// ----------------------------------------------------------------------------

unsigned long Res::Minimum () const
    {
        return minimum;
    }

// ----------------------------------------------------------------------------

unsigned long Res::Users () const
    {
        return users;
    }

// ----------------------------------------------------------------------------

double Res::AvgUsage() const
    {
        const char* where = "Res::AvgUsage";

        if (!valid (className, where))
            return Undefined;

        SimTime now     = CurrentTime();
        SimTime diff    = now - ResetAt();
        double  wSumAvl = wSumAvail
                            + double(avail) * (now - lastReturn).Time();

        if (diff > Epsilon())
            return 1.0 - wSumAvl / (double(limit) * diff.Time());
        else
        {
            Warning ("DivByZero", where, "-1.0 returned");
            return Undefined;
        }
    }

// ----------------------------------------------------------------------------

bool Res::ReleasedAll (Process& p, unsigned long &n)
    {
        const char* where = "Res::ReleasedAll";
        checkProcess (p, where);

        //return (n = resourceDB.AskProvider (res, &p)) != 0;   // Weber: !=
        return (n = resourceDB.AskConsumer (this, &p)) == 0;    // jetzt: ==
    }

// ----------------------------------------------------------------------------
```

```
// ---------------------------------------------------------------------------

void Res::ReleaseAllWithWarning (Process& p, unsigned long wants,
                                             unsigned long holds)
    {
        const char* where = "Res::ReleaseAllWithWarning";
        assert (Valid());
        checkProcess (p, where);

        String info =   "Destruction of process " + p.QuotedName() +
                        ", which currently ";

        if (wants > 0)
        {
            info += "wants " + String(wants) + " resource(s) ";
            resourceDB.DelWaiter (this, &p, wants);
            if (holds > 0)
                info += "and ";
        }
        if (holds > 0)
        {
            info += "holds " + String(holds) + " resource(s) ";
            resourceDB.DelConsumer (this, &p, holds);
        }

        Warning (   info + "of " + QuotedName(),
                    where,
                    "The resource(s) will be released.");

        // update statistik
        SimTime now = CurrentTime();

        wSumAvail += double(Avail()) * (now - lastReturn).Time();
        lastReturn = now;

        avail += holds;
        users++;
    }

// ---------------------------------------------------------------------------

void Res::ConsumerNotFound (Process& p, unsigned long units)
    {
        const char* where = "Res::ConsumerNotFound";
        checkProcess (p, where);

        Warning (   p.QuotedName() + " tried to release " +
                        String(units) + " resource(s) of " + QuotedName() +
                        "but holds none",
                    where);
    }

// ---------------------------------------------------------------------------

void Res::ConsumerReleaseTooMuch (Process&p, unsigned long has,
                                             unsigned long wants)
    {
        const char* where = "Res::ConsumerReleaseTooMuch";
        checkProcess (p, where);

        Warning (   p.QuotedName() + " tried to release " +
                    String(wants) + " resource(s) of " + QuotedName() +
                    "but it holds only " + String(has),
                    where,
                    String(has) + " resource(s) will be released");
    }

// ---------------------------------------------------------------------------

//String Res::Debug ()
//  {
//      const char* where = "Res::Debug";
//
//      strstream ss;
//      long      oflgs = ss.flags(ios::showpoint | ios::fixed | ios::right);
//      int       tw = GetModel().GetExperiment().TimeWidth();
//      int       nw = GetModel().GetExperiment().NameWidth();
//      int       qw = 8;
//      int       w  = nw + pw + tw +2;
//
//      ss  << "Current resource allocation of " << QuotedName() << endl
//          << resetiosflags(ios::left) << setfill('-')
//          << setw (w) << "" << setfill (' ') << endl
//          << setiosflags(ios::left) << setw (nw) << "Entity" << ' '
```

```
//          << resetiosflags(ios::left)
//          << setw (qw) << "Holds" << ' '
//          << setw (qw) << "Wants" << ' '
//          << setw (qw) << "Priority" << ' '
//          << setw (tw) << "Entry in q" << endl
//          << setfill('-')
//          << setw (w) << "" << setfill (' ') << endl;
//
//      for (Process* p = first(); !p->IsNullProcess(); p = p->succ (p))
//          ss  << setiosflags(ios::left) << setw(nw) << p->Name().Left(nw)
//              << resetiosflags(ios::left)
//              << setw(qw+1) << resourceDB.AskConsumer (this, p)
//              << setw(qw+1) << resourceDB.AskWaiter (this, p)
//              << setw(qw+1) << p->GetPriority()
//              << setw(tw+1) << QueueLink::GetQueueLink(*p)->TimeIn()
//              << endl;
//
//      for (QueueLink* ql = first; ql; ql = ql->Next()) {
//          ss  << setiosflags(ios::left) << setw(nw)
//              << ql->GetObject().Name().Left(nw)
//              << resetiosflags(ios::left) << setw(pw+1)
//              << ql->GetObject().GetPriority()
//              << setw(tw+1) << ql->TimeIn() << endl;
//      }
//      ss << ends;
//
//      return Reportable::Debug() + String(ss);
//  }

// ---------------------------------------------------------------------------
```

resdb.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          resdb.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------
//
// Weiterentwicklung von:
//
// Diplomarbeit
//
//          Entwurf und Realisierung eines objektorientierten
//          Simulationspakets in C++
//
// Author
//          Heiko Weber
//
// Beschreibung
//
//          Die Klasse ResourceDB wird benutzt, um Entitys, welche Resourcen
//          belegt haben, zu zu verwalten.
//
// ---------------------------------------------------------------------------

#ifndef  RESOURCEDB_H
#define  RESOURCEDB_H

// ---------------------------------------------------------------------------

class   Avl;
class   ResourceInfoList;
class   Res;
class   DeadlockInfo;
```

```
// ------------------------------------------------------------------------

#include "boolean.h"
#include "experime.h"   // Deadlock-Level

// ------------------------------------------------------------------------

class   ResourceDB
{
public:
                            ResourceDB();
                            ~ResourceDB();

                            // Entity being destroyed
    void                    Destroy (Process*, bool final = false);
        // final == true => Phase, in der pe->model geloescht wird

                            // Consumer => Entity which holds resources
    void                    AddConsumer (Res*, Process*, unsigned long);
    void                    DelConsumer (Res*, Process*, unsigned long);
    unsigned long           AskConsumer (Res* r, Process* p) const;
        // Wieviel belegt p von r?
    unsigned long           AskProvider (Res*& r, Process* p);
        // Belegt p Ressourcen?
        // Wenn ja, in welcher z.B. (muss nicht die erste sein!,
        // da nach Speicheradressen sortiert wird) und wieviele darin
        // wenn 0 zurueckgegeben wird, bleibt r unveraendert!

    unsigned long           AddWaiter (Res* r, Process* p, unsigned long n);
        // Gibt fuer p Anforderungen + Belegungen (inkl. n) von r zurueck
    void                    DelWaiter (Res*, Process*, unsigned long n);
        // Es koennen nur alle Anforderungen auf einmal erfuellt werden
        // (n == wants)
    unsigned long           AskWaiter (Res* r, Process* p) const;
        // Wieviel hat p von r bereits angefordert?

    bool                    SetDeadlockLevel (Experiment::DeadlockLevelT);
                                // true, wenn neuer Level OK
    bool                    DynamicCheckA (Res*, Process*, ostream&);
                                // true bei Deadlock
    bool                    DynamicCheckB (Res*, Process*, ostream&);
                                // true bei Deadlock

    bool                    ResourceDBUsed() const;

    void                    Debug (Res& res, ostream& os, int nw,
                                                          int tw,
                                                          int qw);

private:
    bool                    DeadlockCheck (Res*, Process*, ostream&);
                                // true bei Deadlock

    void                    cleanFlags();
    bool                    search (Process*, DeadlockInfo&);
    bool                    search (Res*, DeadlockInfo&);

                        ResourceDB (const ResourceDB&); // nicht impl.
    ResourceDB&         operator=  (const ResourceDB&); // nicht impl.

    Avl*                dbprov;
    Avl*                dbent;
    ResourceInfoList    *root;
    Experiment::
    DeadlockLevelT      deadlockLevel;
    bool                used;   // DB wurde bereits benutzt
};

// ------------------------------------------------------------------------

#endif //RESOURCEDB_H
```

resdb.cc

```
// ------------------------------------------------------------------------
//
// Datei
```

```
//          resdb.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------
//
// Weiterentwicklung von:
//
// Diplomarbeit
//
//          Entwurf und Realisierung eines objektorientierten
//          Simulationspakets in C++
//
// Author
//          Heiko Weber
//
// Beschreibung
//
//          Die Klasse ResourceDB wird benutzt, um Entitys, welche Resourcen
//          belegt haben, zu zu verwalten.
//
// ---------------------------------------------------------------------------

#include "avl.h"
#include "experimm.h"
#include "process.h"
#include "emessage.h"
#include "res.h"
#include "resdb.h"
#include "ring.h"
#include "str.h"
#include <assert.h>
#include <iostream.h>

// ---------------------------------------------------------------------------
// #define DEBUG_DEADLOCK       // define this for output during dl check
// ---------------------------------------------------------------------------

#ifdef DEBUG_DEADLOCK
#define DEBUG(statement) statement
#else
#define DEBUG(statement)
#endif

// ---------------------------------------------------------------------------

class   ResourceInfo {
private:
    Res                     *provider;              // actually a Res
    Process                 *pentity;               // who 'holds' data
    unsigned long           holds;                  // holds 'n' units
    unsigned long           wants;                  // wants 'n' units
    short                   mask;                   // for deadlock check

public:
                ResourceInfo(Res *rp)
                    : provider(rp), pentity(0), holds(0), wants(0)
                    , mask(0) {}
                ResourceInfo(Res *rp, Process *pe,  unsigned long w = 0,
                                                     unsigned long h = 0)
                    : provider(rp), pentity(pe), holds(h), wants(w)
                    , mask(0) {}
                ResourceInfo(Process *pe)
                    : provider(0), pentity(pe), holds(0), wants(0)
                    , mask(0) {}

    Res*                    Provider(void) const { return provider; }
    Process*                Entity(void) const { return pentity; }

    unsigned long           WantsMore(unsigned long n) { return wants += n; }
    unsigned long           WantsLess(unsigned long n)
                            { assert(n >= wants); return wants -= n; }
    unsigned long           Wants(void) const { return wants; }
```

```
    unsigned long           HoldsMore(unsigned long n) { return holds += n; }
    unsigned long           HoldsLess(unsigned long n)
                            { assert(n >= holds); return holds -= n; }
    unsigned long           Holds(void) const { return holds; }

    void                    ClrMask(void) { mask = 0; }
    short                   Mask(void) { return mask; }
    short                   SetMask(void) { return mask |= 3; }
    short                   SetEntityMask(void) { return mask |= 1; }
    short                   SetProviderMask(void) { return mask |= 2; }

    static AvlCmpResult     cmpProviderEntity(const void*, const void*);
    static AvlCmpResult     cmpEntityProvider(const void*, const void*);
    static AvlCmpResult     cmpProvider(const void*, const void*);
    static AvlCmpResult     cmpEntity(const void*, const void*);

#ifdef   DEBUG_DEADLOCK
    friend ostream         &operator<<(ostream &o, ResourceInfo &r)
    {
        return o << hex   << r.provider << ", "
                 << dec   << r.pentity->Name() << ", "
                 << "H-"  << r.holds << ", "
                 << "W-"  << r.wants << ", "
                 << "C-"  << hex << r.mask << endl;
    }

    static void             print(void *a)
                            { cout << *(ResourceInfo*) a; }
#endif //DEBUG_DEADLOCK
};

// ------------------------------------------------------------------------

AvlCmpResult ResourceInfo::cmpProviderEntity(const void *a, const void *b)
{
    const ResourceInfo *r1 = (const ResourceInfo*) a;
    const ResourceInfo *r2 = (const ResourceInfo*) b;

    if (r1->provider == r2->provider) {
        if (r1->pentity == r2->pentity) return EQUAL;
        return ((unsigned long) r1->pentity < (unsigned long) r2->pentity)
            ? LESS : GREATER;
    }
    return ((unsigned long) r1->provider < (unsigned long) r2->provider)
        ? LESS : GREATER;
}

// ------------------------------------------------------------------------

AvlCmpResult ResourceInfo::cmpEntityProvider(const void *a, const void *b)
{
    const ResourceInfo *r1 = (const ResourceInfo*) a;
    const ResourceInfo *r2 = (const ResourceInfo*) b;

    if (r1->pentity == r2->pentity) {
        if (r1->provider == r2->provider) return EQUAL;
        return ((unsigned long) r1->provider < (unsigned long) r2->provider)
            ? LESS : GREATER;
    }
    return ((unsigned long) r1->pentity < (unsigned long) r2->pentity)
        ? LESS : GREATER;
}

// ------------------------------------------------------------------------

AvlCmpResult ResourceInfo::cmpProvider(const void *a, const void *b)
{
    const ResourceInfo *r1 = (const ResourceInfo*) a;
    const ResourceInfo *r2 = (const ResourceInfo*) b;

    if (r1->provider == r2->provider) return EQUAL;
    return ((unsigned long) r1->provider < (unsigned long) r2->provider)
        ? LESS : GREATER;
}

// ------------------------------------------------------------------------

AvlCmpResult ResourceInfo::cmpEntity(const void *a, const void *b)
{
    const ResourceInfo *r1 = (const ResourceInfo*) a;
    const ResourceInfo *r2 = (const ResourceInfo*) b;

    if (r1->pentity == r2->pentity) return EQUAL;
    return ((unsigned long) r1->pentity < (unsigned long) r2->pentity)
```

```
        ? LESS : GREATER;
}

// ---------------------------------------------------------------------------
// DeadlockInfo
// ---------------------------------------------------------------------------

class   DeadlockInfo {
    String                  msg;
    bool                    flag;

public:
                            DeadlockInfo(void)
                                : flag(false) {}
    void                    Add(ResourceInfo*);
    friend ostream         &operator<<(ostream &o, DeadlockInfo &d)
                            { return o << d.msg; }
};

// ---------------------------------------------------------------------------

void DeadlockInfo::Add(ResourceInfo *sri)
{
    if (flag)
        if (sri->Wants() > 0)
            msg += "which are waited for by '" + sri->Entity()->Name()
                 + "'.\n";
        else
            msg += "which are held by '" + sri->Entity()->Name() + "'.\n";
    else
        if (sri->Holds() > 0)
            msg += sri->Entity()->QuotedName() + " holds resources of '"
                + sri->Provider()->Name() + "',\n";
        else
            msg += sri->Entity()->QuotedName() + " acquires resources of '"
                + sri->Provider()->Name() + "',\n";
    flag = !flag;
}

// ---------------------------------------------------------------------------
// ResourceDB
// ---------------------------------------------------------------------------

class   ResourceInfoList : public Ring<ResourceInfo>
{};

// ---------------------------------------------------------------------------

ResourceDB::ResourceDB(void)
    :   deadlockLevel   (Experiment::DynamicB),
        used            (false)
{
    dbprov = new Avl(ResourceInfo::cmpProviderEntity);
    dbent  = new Avl(ResourceInfo::cmpEntityProvider);
    root   = new ResourceInfoList;
}

// ---------------------------------------------------------------------------

ResourceDB::~ResourceDB(void)
{
    delete root;
    delete dbent;
    delete dbprov;
}

// ---------------------------------------------------------------------------

void ResourceDB::Destroy(Process *pe, bool final)
{   // final == true => Phase, in der pe->model geloescht wird
    const char* where = "ResourceDB::Destroy";
    assert (pe);
    ResourceInfo    ri(pe);
    ResourceInfo   *sri;

    while ((sri = (ResourceInfo*)
            dbent->Search(&ri, ResourceInfo::cmpEntity)) != 0)
    {
        assert (sri->Holds() > 0 || sri->Wants() > 0);
        bool done = false;

        if (!final)
            if (sri->Provider()->Valid())
            {
```

```
                sri->Provider()->ReleaseAllWithWarning
                                      (*pe, sri->Wants(), sri->Holds());
                done = true;
            } else
            {
                CustomErrorMessage msg (
                    "Destruction of process " + pe->QuotedName() + ", which "
                    "wants and/or holds some unit(s) of an invalid resource",
                    where, "resources are released",
                    "", Message::warning);
                ExperimentManager::Instance().Note (msg, *pe);
            }
        if (!done)
        {
            if (!dbprov->Remove(sri)) assert(0);
            if (!dbent->Remove(sri)) assert(0);
            if (!root->Find(sri)) assert(0);
            root->Dequeue();
            delete sri;
        }
    }
}

// ------------------------------------------------------------------------

void ResourceDB::AddConsumer(Res *rp, Process *pe, unsigned long n)
{
    ResourceInfo    ri(rp, pe);
    ResourceInfo   *sri = (ResourceInfo*) dbprov->Search(&ri);

    if (!sri)
    {
        assert (deadlockLevel == Experiment::Off);
        // not found: add a new ResourceInfo the both tree
        sri = new ResourceInfo(rp, pe);
        dbprov->Insert(sri);
        dbent->Insert(sri);
        root->Append(sri);
    } else
    {   // Anforderung loeschen
        if (deadlockLevel != Experiment::Off)
            // Anforderung muss Belegung entsprechen
            assert (sri->Wants() == n);
        sri->WantsLess(sri->Wants());
    }
    sri->HoldsMore(n);
}

// ------------------------------------------------------------------------

void ResourceDB::DelConsumer(Res *rp, Process *pe, unsigned long n)
{   // Consumer werden aufjedenfall in Res:Release abgefragt
    ResourceInfo    ri(rp, pe);
    ResourceInfo   *sri = (ResourceInfo*) dbprov->Search(&ri);

    if (sri)
    {
        if (sri->Holds() >= n)
        {
            sri->HoldsLess(n);
            if  (sri->Holds() == 0 &&
                (sri->Wants() == 0 || deadlockLevel == Experiment::Off))
            {
                if (!dbprov->Remove(sri))   assert(0);
                if (!dbent->Remove(sri))    assert(0);
                if (!root->Find(sri))       assert(0);
                root->Dequeue();
                delete sri;
            }
        } else
            rp->ConsumerReleaseTooMuch(*pe, sri->Holds(), n);
    } else
    {
        // not found: tell the provider
        rp->ConsumerNotFound(*pe, n);
    }
}

// ------------------------------------------------------------------------

unsigned long ResourceDB::AskConsumer(Res *rp, Process *pe) const
{   // wieviel belegt pe in res
    ResourceInfo    ri(rp, pe);
    ResourceInfo   *sri = (ResourceInfo*) dbprov->Search(&ri);
```

```
    if (sri)
        return sri->Holds();
    else
        return 0;
}

// ---------------------------------------------------------------------------

unsigned long ResourceDB::AskProvider (Res*& res, Process *pe)
{   // belegt pe Ressourcen?
    // Wenn ja, in welcher z.B. (muss nicht die erste sein!,
    // da nach Speicheradressen sortiert wird) und wieviele darin
    // wenn 0 zurueckgegeben wird, bleibt res unveraendert!
    // Unerfuellte Anforderungen (wants) bleiben unberuecksichtigt
    ResourceInfo    ri(pe);
    ResourceInfo   *sri = (ResourceInfo*)
                            dbent->Search(&ri, ResourceInfo::cmpEntity);

    if (sri)
    {
        // gefunden:
        res = sri->Provider();
        return sri->Holds();
    } else
    {
        return 0;
    }
}

// ---------------------------------------------------------------------------

unsigned long ResourceDB::AddWaiter (Res *rp, Process *pe, unsigned long n)
{
    // gibt Anforderungen + Belegungen zurueck
    ResourceInfo    ri(rp, pe);
    ResourceInfo*   sri = (ResourceInfo*) dbprov->Search(&ri);

    used = true;

    if (!sri)
    {
        // pe hat noch nichts mit rp zu tun
        if (deadlockLevel != Experiment::Off)
        {   // Waiter registrieren
            sri = new ResourceInfo(rp, pe, n);
            dbprov->Insert(sri);
            dbent->Insert(sri);
            root->Append(sri);
        }
        return n;   // es gab keinen Consumer
    }

    assert (sri->Wants() == 0); // kann nicht zwei mal nacheinander warten
    if (deadlockLevel == Experiment::Off)
        return n + sri->Holds();
    else
        return sri->WantsMore(n) + sri->Holds();
}

// ---------------------------------------------------------------------------

void ResourceDB::DelWaiter(Res *rp, Process *pe, unsigned long n)
{   // es koennen nur alle Anforderungen auf einmal erfuellt werden
    // (n == wants)
    if (deadlockLevel == Experiment::Off)
        return;

    ResourceInfo    ri(rp, pe);
    ResourceInfo   *sri = (ResourceInfo*) dbprov->Search(&ri);

    if (sri)
    {
        if (sri->Wants() >= n) {
            assert (sri->WantsLess(n) == 0);    // kann nur alle bekommen
            if (sri->Holds() == 0 && sri->Wants() == 0)
            {   // wenn keine belegt sind, loeschen:
                if (!dbprov->Remove(sri))   assert(0);
                if (!dbent->Remove(sri))    assert(0);
                if (!root->Find(sri))       assert(0);
                root->Dequeue();
                delete sri;
            }
        } else
```

```
            assert(0);
    } else
    {
        // not found: error
        assert(0);
    }
}

// ---------------------------------------------------------------------------

unsigned long ResourceDB::AskWaiter(Res *rp, Process *pe) const
{   // Wieviel hat pe von r bereits angefordert?
    if (deadlockLevel == Experiment::Off)
        return 0;

    ResourceInfo    ri(rp, pe);
    ResourceInfo   *sri = (ResourceInfo*) dbprov->Search(&ri);

    if (sri)
        return sri->Wants();
    else
        return 0;
}

// ---------------------------------------------------------------------------
// Deadlock checking
// ---------------------------------------------------------------------------

void ResourceDB::cleanFlags(void)
{
    unsigned size = root->Size();

    while (size-- > 0)
        ((ResourceInfo*) root->Next())->ClrMask();
}

// ---------------------------------------------------------------------------

bool ResourceDB::search(Process *pe, DeadlockInfo &dinfo)
{
    ResourceInfo    ri(pe);
    ResourceInfo   *sri, *entry;
    bool            foreward = true;

    entry = (ResourceInfo*) dbent->Search(&ri, ResourceInfo::cmpEntity);
    assert(entry != 0);

    DEBUG (cout << "-----------------" << endl;)
    DEBUG (cout << "entry(E): " << *entry;)
    DEBUG (dbent->Print(ResourceInfo::print);)
    DEBUG (cout << "----------------" << endl;)

    // search entry and foreward, then backwards
    sri = entry;
    do {
        DEBUG (cout << "search(" << pe->Name() << "): " << *sri;)

        if (sri->Holds() > 0) {
            if (sri->Mask()) {
                DEBUG (cout << sri->Entity()->Name() << endl;)
                dinfo.Add(sri);
                return true;
            } else {
                sri->SetEntityMask();
                if (search(sri->Provider(), dinfo)) {
                    DEBUG (cout << sri->Entity()->Name() << endl;)
                    dinfo.Add(sri);
                    return true;
                }
            }
        }
        for (;;) {
            if (foreward) sri = (ResourceInfo*) dbent->Next(sri);
            else          sri = (ResourceInfo*) dbent->Prev(sri);
            if (!sri || sri->Entity() != pe) {
                if (foreward) {
                    foreward = false;
                    sri = entry;
                } else {
                    sri = 0;
                    break;
                }
            } else
                break;
```

```
        }
    } while (sri);

    return false;
}

// ----------------------------------------------------------------------------

bool ResourceDB::search(Res *rp, DeadlockInfo &dinfo)
{
    ResourceInfo    ri(rp);
    ResourceInfo   *sri, *entry;
    bool            foreward = true;

    entry = (ResourceInfo*) dbprov->Search(&ri, ResourceInfo::cmpProvider);
    assert(entry != 0);

    DEBUG (cout << "----------------" << endl;)
    DEBUG (cout << "entry(P): " << *entry;)
    DEBUG (dbprov->Print(ResourceInfo::print);)
    DEBUG (cout << "----------------" << endl;)

    // search entry and foreward, then backwards
    sri = entry;
    do {
        DEBUG (cout << "search(" << hex << rp << "): " << *sri;)

        if (sri->Wants() > 0) {
            if (sri->Mask()) {
                DEBUG (cout << sri->Entity()->Name() << endl;)
                dinfo.Add(sri);
                return true;
            } else {
                sri->SetProviderMask();
                if (search(sri->Entity(), dinfo)) {
                    DEBUG (cout << sri->Entity()->Name() << endl;)
                    dinfo.Add(sri);
                    return true;
                }
            }
        }
        for (;;) {
            if (foreward) sri = (ResourceInfo*) dbprov->Next(sri);
            else          sri = (ResourceInfo*) dbprov->Prev(sri);
            if (!sri || sri->Provider() != rp) {
                if (foreward) {
                    foreward = false;
                    sri = entry;
                } else {
                    sri = 0;
                    break;
                }
            } else
                break;
        }
    } while (sri);

    return false;
}

// ----------------------------------------------------------------------------

bool ResourceDB::DynamicCheckA (Res *rp, Process *pe, ostream& os)
{
    if (deadlockLevel == Experiment::DynamicA)
        return DeadlockCheck (rp, pe, os);
    else
        return false;
}

// ----------------------------------------------------------------------------

bool ResourceDB::DynamicCheckB (Res *rp, Process *pe, ostream& os)
{
    if (deadlockLevel == Experiment::DynamicB)
        return DeadlockCheck (rp, pe, os);
    else
        return false;
}

// ----------------------------------------------------------------------------

bool ResourceDB::DeadlockCheck(Res *rp, Process *pe, ostream& os)
{
```

```
    ResourceInfo    ri(rp, pe);
    ResourceInfo   *sri;
    DeadlockInfo    dinfo;
    bool            deadlockFound = false;

    DEBUG (unsigned size = root->Size();)

    DEBUG (os << "---------------" << endl;)
    DEBUG (while (size-- > 0))
        DEBUG (os << *((ResourceInfo*) root->Next());)
    DEBUG (os << "---------------" << endl;)

    if ((sri = (ResourceInfo*) dbprov->Search(&ri)) != 0)
    {
        cleanFlags();
        sri->SetMask();
        if (search(pe, dinfo))
        {
            DEBUG (os << "Deadlock detect" << endl;)
            os << dinfo;
            deadlockFound = true;
        }
    }

    DEBUG (size = root->Size();)

    DEBUG (os << "---------------" << endl;)
    DEBUG (while (size-- > 0))
        DEBUG (os << *((ResourceInfo*) root->Next());)
    DEBUG (os << "---------------" << endl;)
    return deadlockFound;
}

// ------------------------------------------------------------------------

bool ResourceDB::SetDeadlockLevel (Experiment::DeadlockLevelT dl)
{
    if (dl == deadlockLevel)
        return true;

    if (ResourceDBUsed())
    {
        if (dl != Experiment::Off)
            return false;
        else
        {
            // Deadlock-Ueberwachung ausschalten
            // Nur-Waiter-Eintraege loeschen
            ResourceInfo* sri = root->First();
            for (int i = root->Size(); i > 0; i--)
            {
                if (sri->Holds() == 0)
                {   // Nur-Waiter
                    if (!dbprov->Remove(sri))   assert(0);
                    if (!dbent->Remove(sri))    assert(0);
                    root->Dequeue();
                    delete sri;
                    sri = root->Current();
                } else
                {
                    sri->WantsLess (sri->Wants());
                    sri = root->Next();
                }
            }
        }
    }
    deadlockLevel = dl;
    return true;
}

// ------------------------------------------------------------------------

bool ResourceDB::ResourceDBUsed () const
{
    return used;
}

// ------------------------------------------------------------------------
// noch nicht fertig angepasst:

#include <iomanip.h>

//void ResourceDB::Debug (Res& res, ostream& os, int nw, int tw, int qw)
//  {
```

```
//      const char* where = "ResourceDB::Debug";
//
//      const     pw = 6;
//      long      oflgs = os.flags(ios::showpoint | ios::fixed | ios::right);
//      int       w  = nw + 3*pw + tw +2;

//      os  << "Current resource allocation of " << res.QuotedName()
//          << endl
//          << resetiosflags(ios::left) << setfill('-')
//          << setw (w) << "" << setfill (' ') << endl
//          << setiosflags(ios::left) << setw (nw) << "Entity" << ' '
//          << resetiosflags(ios::left)
//          << setw (qw) << "Holds" << ' '
//          << setw (qw) << "Wants" << ' '
//          << setw (qw) << "Priority" << ' '
//          << setw (tw) << "Entry in q" << endl
//          << setfill('-')
//          << setw (w) << "" << setfill (' ') << endl;

//      for (Process* p = root.first();
//           !p->IsNullProcess();
//           p = p->succ (p))
//      {
//          ss  << setiosflags(ios::left) << setw(nw)
//              << p->Name().Left(nw)
//              << resetiosflags(ios::left)
//              << setw(qw+1) << resourceDB.AskConsumer (this, p)
//              << setw(qw+1) << resourceDB.AskWaiter (this, p)
//              << setw(qw+1) << p->GetPriority()
//              << setw(tw+1) << QueueLink::GetQueueLink(*p)->TimeIn()
//              << endl;
//      }
//
//      for (QueueLink* ql = first; ql; ql = ql->Next())
//      {
//          ss  << setiosflags(ios::left) << setw(nw)
//              << ql->GetObject().Name().Left(nw)
//              << resetiosflags(ios::left) << setw(pw+1)
//              << ql->GetObject().GetPriority()
//              << setw(tw+1) << ql->TimeIn() << endl;
//      }
//      ss << ends;

//      return Reportable::Debug() + String(ss);
//  }

// ---------------------------------------------------------------------------
```

ring.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          ring.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------
//
// Weiterentwicklung von:
//
// Diplomarbeit
//
//          Entwurf und Realisierung eines objektorientierten
//          Simulationspakets in C++
//
// Author
//          Heiko Weber / Mai 1992
```

```
//
// Beschreibung
//
//          Ring - double linked lists
//
// --------------------------------------------------------------------------

#ifndef RING_H
#define RING_H

// --------------------------------------------------------------------------

template<class T>
class Ring_ele {
    public:
        Ring_ele    *prev,
                    *next;
        T           *data;

                     Ring_ele(void) { prev = next = 0; data = 0; }
        Ring_ele*    insertBehind(T* data)
            {
                Ring_ele<T> *r = new Ring_ele<T>;

                r->data = data;

                r->prev = this;
                r->next = next;
                next->prev = r;
                next = r;

                return r;
            };
        Ring_ele*    insertBefore(T* data)
            {
                Ring_ele<T> *r = new Ring_ele<T>;

                r->data = data;

                r->prev     = prev;
                r->next     = this;
                prev->next  = r;
                prev        = r;

                return r;
            };
};

// --------------------------------------------------------------------------

template<class T>
class Ring {
    public:
                     Ring(void)        // Constructor
                        {
                            first = current = 0;
                            count = 0;
                        };
                     Ring(const Ring<T>& r)// Copy-Constructor
                        {
                            first = current = 0;
                            count = 0;
                            Ring_ele<T>* el = r.first;
                            for (int i = r.Size(); i>0; i--)
                            {
                                Append (el->data);
                                el = el->next;
                            }
                        };
        virtual     ~Ring(void)        // Destructor
                        {
                            while (count)
                                Dequeue();
                        };

        unsigned     Size(void) const       // get number of elements
                        {
                            return count;
                        };
        unsigned     Element(T* data) const // search for element,
                        {                   // return Number
                            Ring_ele<T> *dl  = first;
                            unsigned  cnt = 1;
```

```
                        if (!count || !dl) return 0;
                        do {
                            if (dl->data == data)
                                return cnt;
                            ++cnt;
                        } while ((dl = dl->next) != first);

                        return 0;
                    };
        T*          Item(unsigned n)    // get n'th Element
                    {
                        if (n >= count)
                            return 0;
                        First();
                        while (n-- > 0)
                            Next();
                        return Current();
                    };

        T*          Current(void) const     // current element
                    {
                        if (current)
                            return current->data;
                        else
                            return 0;
                    };
        T*          First(void)         // first element
                    {
                        if (first)
                            return (current = first)->data;
                        else
                            return 0;
                    };
        T*          Last(void)          // last element
                    {
                        if (first)
                        {
                            current = first->prev;
                            return first->prev->data;
                        } else
                            return 0;
                    };
        T*          Next(void)          // next element
                    {
                        if (current)
                        {
                            current = current->next;
                            return current->data;
                        } else
                            return 0;
                    };
        T*          Prev(void)          // prev element
                    {
                        if (current)
                        {
                            current = current->prev;
                            return current->data;
                        } else
                            return 0;
                    };

        T*          Dequeue(void)       // remove current and return
                    {
                        T* data;
                        Ring_ele<T> *cur = current;

                        if (!count || cur == 0)
                            return 0;

                        data = cur->data;

                        if (--count == 0)
                            first = current = 0;
                        else {
                            cur->prev->next = current = cur->next;
                            cur->next->prev = cur->prev;
                        }

                        if (cur == first) first = cur->next;
                        delete cur;

                        return data;
                    };
        T*          Pop(void)           // remove first and return
```

```
                        {
                            current = first ;
                            return Dequeue();
                        };
        T*              Enqueue(T* data)  // insert behind current and return
                        {
                            if (first == 0)
                                insertFirst(data);
                            else
                                current = current->insertBehind(data);
                            count++;
                            return data;
                        };
        T*              Insert(T* data)  // insert before current and return
                        {
                            if (first == 0)
                                insertFirst(data);
                            else
                            {
                                if (current == first)
                                    first = current
                                          = current->insertBefore (data);
                                else
                                    current = current->insertBefore (data);
                            }
                            count++;
                            return data;
                        };
        T*              Append(T* data)  // append after last and return
                        {
                            if (first == 0)
                                insertFirst(data);
                            else
                                current = first->prev->insertBehind(data);
                            count++;
                            return data;
                        };
        T*              Push(T* data)   // insert before first and return
                        {
                            Append(data);
                            first = current = first->prev;
                            return data;
                        };

                                    // search for element
        T*              Search(int (*CmpFunc)(const T*, const T*), T* key)
                        {
                            Ring_ele<T> *dl = first;

                            if (!count || !dl) return 0;
                            do {
                                if ((*CmpFunc)(key, dl->data) == 0 ) {
                                    current = dl;
                                    return dl->data ;
                                }
                                dl = dl->next;
                            } while (dl != first) ;

                            return 0;
                        };
        T*              Find(T* data)  // search for element, make current
                        {
                            Ring_ele<T> *dl = _find(data);

                            if (dl)
                                return (current = dl)->data;
                            return 0;
                        };
        T*              Replace(T* from, T* to)  // Replace from, to
                        {
                            Ring_ele<T> *dl = _find(from);

                            if (dl) {
                                T* old = dl->data;
                                dl->data = to;
                                return old;
                            }
                            return 0;
                        };
        T*              Remove(T* data)  // Remove element from ring
                        {
                            if (Find(data)) // => current = data
                                return Dequeue();
                            else
```

```
                                return 0;
                        };
    private:
        Ring_ele<T> *first,
                    *current;
        unsigned     count;

        Ring_ele<T>* insertFirst(T* data)
                        {
                            Ring_ele<T> *cur = new Ring_ele<T>;

                            current = first = cur;
                            cur->prev = cur->next = cur;
                            cur->data = data;

                            return cur;
                        };
        Ring_ele<T>* _find (T* data) const
                        {
                            Ring_ele<T> *dl = first;

                            if (!count || !dl) return 0;
                            do {
                                if (dl->data == data)
                                    return dl;
                            } while ((dl = dl->next) != first);

                            return 0;
                        };

};

// ---------------------------------------------------------------------------

#endif // RING_H
```

schedula.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          schedula.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#ifndef SCHEDULABLE_H
#define SCHEDULABLE_H

// ---------------------------------------------------------------------------

#include "dyobject.h"
#include "boolean.h"
#include "str.h"

// ---------------------------------------------------------------------------

class Event;
class Entity;
class Process;
class EventNote;

// ---------------------------------------------------------------------------

class Schedulable : public DynamicalObject
    /*  Schedulable soll Oberklasse fuer Entity (und Event) werden
        hier werden Methoden bereitgestellt, ueber die die Nachfolger
```

```
        ermittelt werden koennen sowie Methoden, die fuer Events, Entities
        und Prozesse identisch sind (ReSchedule, Cancel, IsScheduled und
        ScheduledAt). */
    /*  Der Scheduler ist als friend deklariert, damit er die Verknuepfung
        mit einer EventNote vornehmen kann.
    */
{
        friend class Scheduler;

        Schedulable& operator= (const Schedulable&); // nicht implementiert

    public:
                    Schedulable (   Model&  owner,
                            const   String& name = "",
                                    bool    showInTrace = true);
                    Schedulable (const Schedulable&);
                        /*  Copy-Konstruktor: das neue Objekt erhaelt
                            lediglich die 'DynamicalObject'-Anteile seiner
                            'Kopiervorlage'. Ansonsten ist es ein neues
                            Objekt.
                        */
        virtual    ~Schedulable ();

        bool            IsNull () const;
                            // Abfrage, ob das Objekt ein Pseudo-Objekt ist

        bool            IsCurrent () const;
                            // Abfrage, ob das Objekt das aktuelle ist
        bool            IsScheduled () const;
                            // Abfrage, ob das Objekt bereits vorgemerkt ist
        SimTime         ScheduledAt () const;   // DESMO: EvTime ()
                            // Zeitpunkt, zu dem das Objekt vorgemerkt ist

        Schedulable&    Next() const;
                            /*  Versucht, das naechste vorgemerkte Entity
                                zu liefern sonst das naechste Event
                                oder NullEvent */
        Event&          NextEvent () const;
                            // Das naechste vorgemerkte Ereignis
        Entity&         NextEntity () const;
                            // Das naechste vorgemerkte Entity
        Process&        NextProcess () const;
                            // Der naechste vorgemerkte Prozess

        void            ReSchedule (SimTime dt);
                            //  Verschieben auf der Ereignisliste auf now + dt
                            /*  Mit 'ReSchedule' kann fuer einen Prozess keine
                                Verdraengung erreicht werden (dt == Now()).
                                Hierfuer muss bei Prozessen 'ReActivate'
                                verwendet werden.*/
        void            Cancel ();
                            // Entfernen von der Ereignisliste

        virtual void    Rename (const String& newName);
                            /*  erweitert NamedObject::Rename um die
                                Namensnumerierung */
        String          ClassName () const;

    private:
        EventNote*  eventNote;
};

// ------------------------------------------------------------------------------

#endif  // SCHEDULABLE_H
```

schedula.cc

```
// ------------------------------------------------------------------------------
//
// Datei
//          schedula.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
```

```
//
// Autor
//         Thomas Schniewind
//
// Datum
//         8.3.1998
//
// -----------------------------------------------------------------------------

#include "schedula.h"

#include "emessage.h"
#include "msgsched.h"
#include "eventlis.h"   // fuer eventNote->Time()
#include "experime.h"
#include "experimm.h"
#include "model.h"
#include "process.h"    // fuer IsNull()
#include "namecat.h"
#include "schedule.h"
#include "simtime.h"

#include <assert.h>

// -----------------------------------------------------------------------------

static const char* className = "Schedulable";

// -----------------------------------------------------------------------------

Schedulable::Schedulable (Model& owner, const String& name, bool showInTrace)
    :   DynamicalObject     (owner,
                             ExperimentManager::Instance().
                                GetNameCatalog (owner).AddNumberTo (name),
                             showInTrace),
        eventNote           (0)
    {
    }

// -----------------------------------------------------------------------------

Schedulable::Schedulable (const Schedulable& s)
    :   DynamicalObject     (s),
        eventNote           (0)
    {}

// -----------------------------------------------------------------------------

Schedulable::~Schedulable ()
    {
        const char* where = "Schedulable::~Schedulable";

        if (IsScheduled())
        {
            if (!IsGarbage())
            {
                Warning (   "attempt to delete the scheduled " + ClassName()
                            + ' ' + QuotedName(),
                            where,
                            ClassName() + " is canceld before");
                Cancel();
            }
            else
                //  Loeschung im Rahmen der Garbage-Collection
                //  Keine Warnung ausgeben
                ExperimentManager::Instance().GetScheduler (*this).
                    Cancel (*this);
        }
    }

// -----------------------------------------------------------------------------

void Schedulable::Rename (const String& name)
    {
        DynamicalObject::Rename (ExperimentManager::Instance().
                                GetNameCatalog (*this).AddNumberTo (name));
    }

// -----------------------------------------------------------------------------

void Schedulable::ReSchedule (SimTime dt)
    // verschieben auf der Ereignisliste auf now + dt
    {
        const char* where = "Schedulable::ReSchedule";
```

```
        // this pruefen
        if (!valid (className, where))
            return;
        if (!IsScheduled())
        {
            Warning (  "attempt to reschedule a not scheduled " + ClassName(),
                       where, "ignored");
            return;
        }

        if (dt < 0.0 && dt != NOW())
        {
            Warning (  "negativ dt [" + String(dt.Time()) + ']',
                       where, "0.0 is used");
            dt = 0.0;
        }

        if (TraceIsOn())
            SendMessage (TrcReSchedule (dt, *this));

        Scheduler& scheduler = ExperimentManager::Instance().
                                   GetScheduler(*this);
        scheduler.ReSchedule (dt, *this);
    }

// ------------------------------------------------------------------------

void Schedulable::Cancel ()
    // Entfernen von der Ereignisliste
    {
        const char* where = "Schedulable::Cancel";

        if (!valid (className, where))
            return;

        if (!IsScheduled())
        {
            Warning (   "attempt to cancel a not scheduled " + ClassName(),
                        where, "ignored");
            return;
        }

        if (TraceIsOn())
            SendMessage (TrcCancel (*this));

        Scheduler& scheduler = ExperimentManager::Instance().
                                   GetScheduler (*this);
        scheduler.Cancel (*this);
    }

// ------------------------------------------------------------------------

bool Schedulable::IsScheduled () const
    // Abfrage, ob das Objekt bereits vorgemerkt ist
    {
        return eventNote != 0;
    }

// ------------------------------------------------------------------------

SimTime Schedulable::ScheduledAt () const   // DESMO: EvTime ()
    // Zeitpunkt, zu dem das Objekt vorgemerkt ist
    {
        const char* where = "Schedulable::ScheduledAt";

        if (!valid (className, where))
            return -1.0;

        if (!IsScheduled())
        {
            Warning (   "object is not scheduled",
                        where, "-1.0 is returned");
            return -1.0;
        }
        assert (eventNote); // <==> IsScheduled();

        return eventNote->Time();
    }

// ------------------------------------------------------------------------

bool Schedulable::IsCurrent () const
    // Abfrage, ob das Objekt bereits vorgemerkt ist
```

```
    {
        return this == &CurrentEntity() || this == &CurrentEvent();
    }

// ---------------------------------------------------------------------------

bool Schedulable::IsNull () const
    // Abfrage, ob das Objekt bereits vorgemerkt ist
    {
        return this == &NullEntity() || this == &NullEvent();
    }

// ---------------------------------------------------------------------------

Schedulable& Schedulable::Next () const
    // Das naechste vorgemerkte Schedulable (vorzugsweise Entity)
    {
        const char* where = "Schedulable::Next";

        if (!IsScheduled() && !IsCurrent())
        {
            Warning (   "attempt to call 'Next' on a not scheduled "
                        + ClassName(),
                        where, "NullEvent is returned");
            return NullEvent();
        }

        Scheduler& scheduler = ExperimentManager::Instance().
                                   GetScheduler(*this);
        return scheduler.Next (*this);
    }

// ---------------------------------------------------------------------------

Event& Schedulable::NextEvent () const
    // Das naechste vorgemerkte Ereignis
    {
        const char* where = "Schedulable::NextEvent";

        if (!IsScheduled() && !IsCurrent())
        {
            Warning (   "attempt to call 'NextEvent' on a not scheduled "
                        + ClassName(),
                        where, "NullEvent is returned");
            return NullEvent();
        }

        Scheduler& scheduler = ExperimentManager::Instance().
                                   GetScheduler(*this);
        return scheduler.NextEvent (*this);
    }

// ---------------------------------------------------------------------------

Entity& Schedulable::NextEntity () const    // DESMO: NextEv ()
    // Das naechste vorgemerkte Entity
    {
        const char* where = "Schedulable::NextEntity";

        if (!IsScheduled() && !IsCurrent())
        {
            Warning (   "attempt to call 'NextEntity' on a not scheduled "
                        + ClassName(),
                        where, "NullEntity is returned");
            return NullEntity();
        }

        Scheduler& scheduler = ExperimentManager::Instance().
                                   GetScheduler(*this);
        return scheduler.NextEntity (*this);
    }

// ---------------------------------------------------------------------------

Process& Schedulable::NextProcess () const
    // Der naechste vorgemerkte Prozess
    {
        const char* where = "Schedulable::NextProcess";

        if (!IsScheduled() && !IsCurrent())
        {
            Warning (   "attempt to call 'NextProcess' on a not scheduled "
                        + ClassName(),
                        where, "NullProcess is returned");
```

```
            return NullProcess();
        }

        Scheduler& scheduler = ExperimentManager::Instance().
                                    GetScheduler(*this);
        return scheduler.NextProcess (*this);
    }

// ------------------------------------------------------------------------------

String Schedulable::ClassName () const
    {
        return className;
    }

// ------------------------------------------------------------------------------
```

schedule.h

```
// ------------------------------------------------------------------------------
//
// Datei
//          schedule.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ------------------------------------------------------------------------------

#ifndef SCHEDULER_H
#define SCHEDULER_H

// ------------------------------------------------------------------------------

class DynamicalObject;
class DynObjList;
class EventList;
class Schedulable;
class Entity;
class Event;
class Process;
class Model;

// ------------------------------------------------------------------------------

#include "boolean.h"
#include "entity.h"
#include "simclock.h"
#include "simtime.h"
#include "str.h"

// ------------------------------------------------------------------------------

class Scheduler
{
    /*  Achtung: Ein Prozess, der ereignisorientiert vorgemerkt
        ist, ist auf jedenfall 'Scheduled', und kann somit nicht
        aktiviert werden.
        Achtung: Die Semantik von 'ReSchedule'
        haengt bei Prozessen von der Art der Vormerkung ab
        (ereignis- oder prozessorientiert).
    */

    public:
        Scheduler (SimTime& epsilon);
        ~Scheduler ();

        void            ChangeEventList (const EventList& prototyp);
                            // der Prototyp wird nach dem Aufruf vom Scheduler
```

```
                        // nicht mehr benoetigt

    bool            ProcessNextEventNote();
                        // arbeitet nur die naechste Ereignisnotiz ab;
                        // liefert true, wenn eine Notiz abgearbeitet
                        // werden konnte

    SimTime         Epsilon () const;
    SimTime         CurrentTime () const;
    Event&          CurrentEvent () const;
    Entity&         CurrentEntity () const;
    Process&        CurrentProcess () const;
    Model&          CurrentModel () const;

    SimClock&       GetSimClock ();
    void            SetCurrentTime (SimTime t);
    void            SetCurrentModel (Model&);

    Schedulable&    Next        (const Schedulable& s) const;
    Event&          NextEvent   (const Schedulable& s) const;
    Entity&         NextEntity  (const Schedulable& s) const;
    Process&        NextProcess (const Schedulable& s) const;

    Schedulable&    Prev        (const Schedulable& s) const;
    Event&          PrevEvent   (const Schedulable& s) const;
    Entity&         PrevEntity  (const Schedulable& s) const;
    Process&        PrevProcess (const Schedulable& s) const;

    void            Schedule (const SimTime& dt,
                              Event& ev,
                              Entity& en = ModelComponent::NullEntity());
        // ereignisorientiert
        // Vorbedingung: weder ev noch en duerfen vorgemrkt sein

    void            Schedule (const SimTime& dt,
                                    Process& p);
        // prozessorientiert
        // Vorbedingung: p darf nicht vorgemrkt sein

    void            ReSchedule (const SimTime& dt, Schedulable& s);
        // ereignisorientiert
        // Vorbedingung: s muss vorgemrkt sein

    void            ScheduleBefore (Schedulable& before,
                                    Event&       ev,
                                    Entity&      en);
        // Vorbedingung: before muss vorgemerkt sein,
        //               ev und en duerfen es nicht

    void            ScheduleAfter (Schedulable& after,
                                   Event&       ev,
                                   Entity&      en);
         // Vorbedingung: after muss vorgemerkt sein,
         //               ev und en duerfen es nicht

    void            Cancel (Schedulable& s);
        // Vorbedungung: s muss vorgemerkt sein
        //               s darf nicht current sein
        // entfernt s von der Ereignisliste

    void            Passivate (Process& p);
        /*  Vorbedingung: p != NullProcess
            p muss der gerade laufende Prozess sein. */

    void            InitCurrentObjects();
        /*  Initialisiert currentEvent, currentEntity, currentProcess
            auf die entspr. NullObjekte.
            (nur fuer DefaultExperiment) */

    void            Terminate (DynamicalObject& d);
        /*  merkt d zum Loeschen vor */

    void            PrepareDeletionOf (Model& m);
        /*  setzt currents auf nullobjekte, falls sie zu m gehoeren */

private:
    void            preemptCurrent (EventNote*);
        /*  Vorbedingung: currentProcess != NullProcess
            verdraengt currentProcess mit den Objekten in EventNote
        */

    void            Debug ();
        /*  Gibt Informationen in die Debug-Ausgabe aus */
```

```
        void            Debug (EventNote* note, ostream& os,
                                int timeWidth,
                                int nameWidth,
                                int priorityWidth,
                                int queueOffset);
            /*  Gibt eine Ereignisnotiz in die Debug-Ausgabe aus */

        EventList*      eventList;
        DynObjList*     dynObjToDelete;
        SimClock        simClock;
        Event*          currentEvent;
        Entity*         currentEntity;
        Process*        currentProcess;
        Model*          currentModel;
};

// ---------------------------------------------------------------------------

#endif  // SCHEDULER_H
```

schedule.cc

```
// ---------------------------------------------------------------------------
//
// Datei
//          schedule.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

//#define NDEBUG  // Zusicherungen ueber assert deaktivieren

#include "schedule.h"

#include "entity.h"
#include "event.h"
#include "experime.h"
#include "experimm.h"
#include "linevlis.h"
#include "messagem.h"
#include "msgtypes.h"
#include "process.h"
#include "pimpl.h"      // Zugriff auf Prozess-Implementierung fuer Transfer
#include "simclock.h"
#include "model.h"
#include "coroutin.h"   // fuer Aktivierung des Hauptprogramms

#include "qlink.h"      // fuer Debug
#include "qimpl.h"      // fuer Debug

#include "schedula.h"

#include "ring.h"

#include <iostream.h>
#include <iomanip.h>
#include <assert.h>
#include <stdlib.h> // fuer abort()

// ---------------------------------------------------------------------------

class DynObjList : public Ring<DynamicalObject> {};

// ---------------------------------------------------------------------------

Scheduler::Scheduler (SimTime& epsilon)
    :   eventList       (new LinearEventList),
```

```
        dynObjToDelete  (new DynObjList),
        simClock        (epsilon),
        currentEvent    (ExperimentManager::Instance().GetNullEvent()),
        currentEntity   (ExperimentManager::Instance().GetNullEntity()),
        currentProcess  (ExperimentManager::Instance().GetNullProcess()),
        currentModel    (ExperimentManager::Instance().GetNullModel())
    {}

// ---------------------------------------------------------------------------

Scheduler::~Scheduler ()
    {
        EventNote*  note = eventList->FirstEventNote();

        while (note)
        {
            if (note->event)
                note->event->eventNote = 0;
            if (note->entity)
                note->entity->eventNote = 0;
            note = eventList->NextEventNote (note);
        }
        delete eventList;
        delete dynObjToDelete;
    }

// ---------------------------------------------------------------------------

void Scheduler::ChangeEventList (const EventList& prototype)
    {
        EventNote*  note;
        EventList* tempList = prototype.Clone (*eventList);
        if (tempList)
        {
            Debug();
            delete eventList;
            eventList = tempList;

            // Verbindung zu den Schedulable aktualisieren:
            note = eventList->FirstEventNote();
            while (note)
            {
                if (note->event && !note->event->IsNullEvent())
                    note->event->eventNote = note;
                if (note->entity && !note->entity->IsNullEntity())
                    note->entity->eventNote = note;
                note = eventList->NextEventNote (note);
            }
            Debug();
        }
    }

// ---------------------------------------------------------------------------

bool Scheduler::ProcessNextEventNote ()
    // Wichtig: Nach Verlassen von ProcessNextEventNote muss der Scheduler und
    // die Ereignisliste in einem konsistenten Zustand sein!!!
    {
        if (EventNote* note = eventList->FirstEventNote ())
        {   // Liste bleibt noch unveraendert!
            // Vorkerungen treffen um current und First zu sperren
            // ...

            // Die Uhr kuemmert sich um Benachrichtigungen von Objekten,
            // die ueber eine Zeitfortschreibung informiert werden wollen.
            // Die Zeit wird erst nach der Benachrichtigung gesetzt
            simClock.SetTime (note->Time());

            // ...
            // Sperrung aufheben
            assert (note == eventList->FirstEventNote());
            // darf sich nicht geaendert haben

            // jetzt erst die erste Notiz entfernen
            eventList->Remove (note);

            currentEvent = note->event;
            // muss immer ein gueltiges Ereignis sein (evtl. NullEvent)
            assert (currentEvent);

            currentEntity = note->entity;
            // muss immer ein gueltiges Entity sein (evtl. NullEntity)
            assert (currentEntity);
```

```
            currentEvent->eventNote =       // Verbindungen loesen
            currentEntity->eventNote = 0;

            delete note;                    // wird nun nicht mehr gebraucht

            if (currentEntity->IsProcess())
            {
                currentProcess = (Process*)currentEntity;
                currentModel   = &currentEntity->GetModel();
            }
            else
            {
                currentProcess = &ModelComponent::NullProcess();
                currentModel   = &currentEvent->GetModel();
            }

            Debug();

            // evtl. terminierte Objekte loeschen:
            DynamicalObject* d = dynObjToDelete->First();
            for (int i=1; i<= dynObjToDelete->Size(); i++)
            {
                if (d != currentEvent && d != currentEntity)
                {
                    delete dynObjToDelete->Dequeue();
                    d = dynObjToDelete->Current();
                }
                else
                    d = dynObjToDelete->Next();
            }

            assert (    !currentProcess->IsNullProcess()
                    ||  !currentEvent  ->IsNullEvent  ());
            // hier gilt:
            //      currentEvent    != 0;
            //      (currentEntity  == 0) => externes Ereignis
            //      (currentProcess != 0) => currentEvent == NullEvent
            //      (currentEntity  != 0 && !currentEntity->IsProcess())
            //                            => !currentEvent->IsNullEvent()

            // MainCoroutine hier immer aktiv!
            if (currentProcess == &ModelComponent::NullProcess())
            {
                // ereignisorientiert
                currentModel = &currentEvent->GetModel();
                currentEvent->EventRoutine (*currentEntity);
                if (currentEvent->CheckDeleteOnTermination())
                    Terminate (*currentEvent);
            }
            else
            {
                // prozessorientiert
                currentModel = &currentProcess->GetModel();
                currentProcess->coroutine.Transfer();
            }

            // MainCoroutine hier immer aktiv!

            return true;    // Ereignisliste war noch nicht leer
        }
        else
            return false;
    }

// ------------------------------------------------------------------------------

SimTime Scheduler::Epsilon () const
    {
        return simClock.Epsilon();
    }

// ------------------------------------------------------------------------------

SimTime Scheduler::CurrentTime () const
    {
        return simClock.Time();
    }

// ------------------------------------------------------------------------------

Event& Scheduler::CurrentEvent () const
    {
        assert (currentEvent);
        return *currentEvent;
```

```
    }

// ---------------------------------------------------------------------------

Entity& Scheduler::CurrentEntity () const
    {
        assert (currentEntity);
        return *currentEntity;
    }

// ---------------------------------------------------------------------------

Process& Scheduler::CurrentProcess () const
    {
        assert (currentProcess);
        return *currentProcess;
    }

// ---------------------------------------------------------------------------

Model& Scheduler::CurrentModel () const
    {
        assert (currentModel);
        return *currentModel;
    }

// ---------------------------------------------------------------------------

SimClock& Scheduler::GetSimClock ()
    {
        return simClock;
    }

// ---------------------------------------------------------------------------

void Scheduler::SetCurrentTime (SimTime t)
    {
        simClock.SetTime (t);
    }

// ---------------------------------------------------------------------------

void Scheduler::SetCurrentModel (Model& m)
    {
        currentModel = &m;
    }

// ---------------------------------------------------------------------------

Schedulable& Scheduler::Next (const Schedulable& s) const
    {
        EventNote* note = s.eventNote;

        if (note)
            note = eventList->NextEventNote (note);
        else
        {
            if (&s == currentEvent || &s == currentEntity)
                note = eventList->FirstEventNote ();
            else
                assert (0); // note = 0;
        }

        if (note)
            if (note->entity)   // muss immer erfuellt sein!
                if (note->entity->IsNullEntity())
                    if (note->event)    // muss immer erfuellt sein!
                        return *note->event;
                    else
                        return ModelComponent::NullEvent();
                else
                    return *note->entity;
            else
                return ModelComponent::NullEvent(); // Kein weiteres Event
        else
            return ModelComponent::NullEvent(); // Keine weitere Notiz
    }

// ---------------------------------------------------------------------------

Event& Scheduler::NextEvent (const Schedulable& s) const
    {
        EventNote* note = s.eventNote;
```

```
        if (note)
            note = eventList->NextEventNote (note);
        else
        {
            if (&s == currentEvent || &s == currentEntity)
                note = eventList->FirstEventNote ();
            else
                assert (0); // note = 0;
        }

        if (note)
            if (note->event)    // muss immer erfuellt sein!
                return *note->event;
            else
                return ModelComponent::NullEvent(); // Kein weiteres Entity
        else
            return ModelComponent::NullEvent(); // Keine weitere Notiz
    }

// --------------------------------------------------------------------------

Entity& Scheduler::NextEntity (const Schedulable& s) const
    {
        EventNote* note = s.eventNote;

        if (note)
            note = eventList->NextEventNote (note);
        else
        {
            if (&s == currentEvent || &s == currentEntity)
                note = eventList->FirstEventNote ();
            else
                assert (0); // note = 0;
        }

        if (note)
            if (note->entity)   // muss immer erfuellt sein!
                return *note->entity;
            else
                return ModelComponent::NullEntity();// Kein weiteres Entity
        else
            return ModelComponent::NullEntity();    // Keine weitere Notiz
    }

// --------------------------------------------------------------------------

Process& Scheduler::NextProcess (const Schedulable& s) const
    {
        EventNote* note = s.eventNote;

        if (note)
            note = eventList->NextEventNote (note);
        else
        {
            if (&s == currentEvent || &s == currentEntity)
                note = eventList->FirstEventNote ();
            else
                assert (0); // note = 0;
        }

        if (note)
            if (note->entity && note->entity->IsProcess())
                return (Process&)*note->entity;
            else
                return ModelComponent::NullProcess();// Kein weiterer Process
        else
            return ModelComponent::NullProcess();    // Keine weitere Notiz
    }

// --------------------------------------------------------------------------

Schedulable& Scheduler::Prev (const Schedulable& s) const
    {
        EventNote* note = s.eventNote;

        if (note)
        {
            if (note == eventList->FirstEventNote ())
                if (currentEntity->IsNullEntity())
                    return *currentEvent;
                else
                    return *currentEntity;
            else
                note = eventList->PrevEventNote (note);
```

```
        }

        if (note)
            if (note->entity)   // muss immer erfuellt sein!
                if (note->entity->IsNullEntity())
                    if (note->event)    // muss immer erfuellt sein!
                        return *note->event;
                    else
                        return ModelComponent::NullEvent();
                else
                    return *note->entity;
            else
                return ModelComponent::NullEvent(); // Kein weiteres Event
        else
            return ModelComponent::NullEvent();     // Keine weitere Notiz
    }

// ---------------------------------------------------------------------------

Event& Scheduler::PrevEvent (const Schedulable& s) const
    {
        if (EventNote* note = s.eventNote)
            if (note == eventList->FirstEventNote())
                return *currentEvent;
            else
                if ((note = eventList->PrevEventNote (note)) != 0)
                    if (note->event)    // muss immer erfuellt sein!
                        return *note->event;
                    else
                        return ModelComponent::NullEvent(); // Kein Event
                else
                    return ModelComponent::NullEvent(); // Keine weitere Notiz
        else
            return ModelComponent::NullEvent(); // s ist nicht vorgemerkt!
    }

// ---------------------------------------------------------------------------

Entity& Scheduler::PrevEntity (const Schedulable& s) const
    {
        if (EventNote* note = s.eventNote)
            if (note == eventList->FirstEventNote())
                return *currentEntity;
            else
                if ((note = eventList->PrevEventNote (note)) != 0)
                    if (note->entity)   // muss immer erfuellt sein!
                        return *note->entity;
                    else
                        return ModelComponent::NullEntity();// Kein Entity
                else
                    return ModelComponent::NullEntity(); // Keine Notiz
        else
            return ModelComponent::NullEntity();    // s ist nicht vorgemerkt!
    }

// ---------------------------------------------------------------------------

Process& Scheduler::PrevProcess (const Schedulable& s) const
    {
        if (EventNote* note = s.eventNote)
            if (note == eventList->FirstEventNote())
                return *currentProcess;
            else
                if ((note = eventList->PrevEventNote (note)) != 0)
                    if (note->entity && note->entity->IsProcess())
                        return (Process&)*note->entity;
                    else
                        return ModelComponent::NullProcess();// Kein Process
                else
                    return ModelComponent::NullProcess();   // Keine Notiz
        else
            return ModelComponent::NullProcess();   // s ist nicht vorgemerkt!
    }


// ---------------------------------------------------------------------------
// Ereignis- und Prozessorientierte Methoden:
// ---------------------------------------------------------------------------

void Scheduler::Schedule (const SimTime& dt, Event& event, Entity& entity)
    // event != NullEvent() => Ereignisorientiertes Schedule
    // event == NullEvent() => Prozessorientiertes Schedule
    {
        // Vorbedingungen:
```

```
        assert (currentEvent && currentEntity && currentProcess);
        assert ((dt >= 0.0) || (dt == DynamicalObject::NOW()));
        assert (!event.IsScheduled());  // darf nicht vorgemerkt sein
        assert (!entity.IsScheduled()); // dto.
               // (NullEntities duerfen nie vorgemerkt sein!)
        assert (    !event.IsNullEvent()
                || (entity.IsProcess() && !entity.IsNullEntity()));

        // sowohl entity als auch event duerfen == current sein

        SimTime time = simClock.Time() + dt;

        if (dt == DynamicalObject::NOW())
            time = simClock.Time();

        EventNote* note = eventList->NewEventNote (time, &event, &entity);
        // Ereignis und Entity mit EventNote verknuepfen:
        if (!event.IsNullEvent())
            event.eventNote  = note;
        if (!entity.IsNullEntity ())
            entity.eventNote = note;

        if (dt == DynamicalObject::NOW())
            if (currentProcess->IsNullProcess())
            {
                // Ereignisse koennen nicht verdraengt werden;
                EventNote* n = eventList->InsertAsFirst (note);
                assert (n);
                Debug();
            }
            else
            {
                preemptCurrent (note);
                //delete note;  // wird nun nicht mehr benoetigt
                // currentProcess verdraengen:
                //preemptCurrent (event, entity);
            }
        else
        {
            EventNote* n = eventList->Insert (note);
            assert (n);
            Debug();
        }
    }

// ------------------------------------------------------------------------------

void Scheduler::Schedule (const SimTime& dt, Process& process)
    {
        Schedule (dt, ModelComponent::NullEvent(), process);
    }

// ------------------------------------------------------------------------------

void Scheduler::ReSchedule (const SimTime& dt, Schedulable& s)
    // Vorbedingung: s muss vorgemrkt sein, s kann auch Process sein
    {
        // Vorbedingungen:
        assert (currentEvent && currentEntity && currentProcess);
        assert ((dt >= 0.0) || (dt == DynamicalObject::NOW()));
        assert (s.IsScheduled());

        EventNote* note = s.eventNote;
        assert (note);
        eventList->Remove (note);
        // note wird weiterverwendet

        if (dt == DynamicalObject::NOW())
        {
            note->time = simClock.Time();
            if (currentProcess->IsNullProcess())
            {
                EventNote* n = eventList->InsertAsFirst (note);
                assert (n);
                Debug();
            }
            else
                preemptCurrent (note);
        }
        else
        {
            note->time = simClock.Time() + dt;
            EventNote* n = eventList->Insert (note);
            assert (n);
```

```
            Debug();
        }
    }

// ------------------------------------------------------------------------

void Scheduler::ScheduleBefore (Schedulable& before, Event& ev, Entity& en)
    // Vorbedingung: before muss vorgemerkt oder current sein,
    // ev und en duerfen es nicht
    // before darf nur current sein, wenn es auch ein Prozess ist
    {
        // Vorbedingungen:
        assert (currentEvent && currentEntity && currentProcess);
        assert (!ev.IsScheduled());
        assert (!en.IsScheduled());
        assert (!ev.IsNullEvent() || (ev.IsNullEvent() && en.IsProcess()));


        if (before.IsCurrent())
        {
            assert (!currentProcess->IsNullProcess());
                // Verdraengung nur wenn Prozess aktiv
            EventNote* note = eventList->NewEventNote
                                         (simClock.Time(), &ev, &en);
            // ev und en mit note verknuepfen
            if (!ev.IsNullEvent())
                ev.eventNote = note;
            if (!en.IsNullEntity())
                en.eventNote = note;
            preemptCurrent (note);
        }
        else
        {
            assert (before.IsScheduled());  // => before != NullObject
            assert (before.eventNote);

            EventNote* note = eventList->NewEventNote
                                        (before.eventNote->time, &ev, &en);
            // ev und en mit note verknuepfen
            if (!ev.IsNullEvent())
                ev.eventNote = note;
            if (!en.IsNullEntity())
                en.eventNote = note;

            EventNote* n = eventList->InsertBefore (before.eventNote, note);
            assert (n);
            Debug();
        }
    }

// ------------------------------------------------------------------------

void Scheduler::ScheduleAfter (Schedulable& after, Event& ev, Entity& en)
    // Vorbedingung: after muss vorgemerkt oder current sein,
    //               ev und en duerfen es nicht
    {
        // Vorbedingungen:
        assert (currentEvent && currentEntity && currentProcess);
        assert (!ev.IsScheduled());
        assert (!en.IsScheduled());
        assert (!ev.IsNullEvent() || (ev.IsNullEvent() && en.IsProcess()));

        if (after.IsCurrent())
        {
            EventNote* note = eventList->NewEventNote
                                        (simClock.Time(), &ev, &en);
            // ev und en mit note verknuepfen
            if (!ev.IsNullEvent())
                ev.eventNote = note;
            if (!en.IsNullEntity())
                en.eventNote = note;
            eventList->InsertAsFirst (note);
            Debug();
        }
        else
        {
            assert (after.IsScheduled());   // => after != NullObject
            assert (after.eventNote);

            EventNote* note = eventList->NewEventNote
                                       (after.eventNote->Time(), &ev, &en);
            // ev und en mit note verknuepfen
            if (!ev.IsNullEvent())
                ev.eventNote = note;
```

```
            if (!en.IsNullEntity())
                en.eventNote = note;

            EventNote* n = eventList->InsertAfter (after.eventNote, note);
            assert (n);
            Debug();
        }
    }

// ---------------------------------------------------------------------------

void Scheduler::Cancel (Schedulable& s)
    // Vorbedungung: s muss vorgemerkt sein
    {
        assert (s.IsScheduled());
        EventNote* note = s.eventNote;
        assert (note);
        assert (note->event && note->entity);
        assert (note->event == &s || note->entity == &s);

        eventList->Remove (note);
        note->event->eventNote = 0;
        note->entity->eventNote = 0;

        delete note;
        Debug();
    }

// ---------------------------------------------------------------------------
// Prozessorientierte Methoden:
// ---------------------------------------------------------------------------

void Scheduler::Passivate (Process& p)
    {
        assert (&p == currentProcess);
        assert (!p.IsNullProcess());

        Coroutine::MainCoroutine()->Transfer();
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

void Scheduler::InitCurrentObjects ()
    {
        currentEvent    = &ModelComponent::NullEvent();
        currentEntity   =
        currentProcess  = &ModelComponent::NullProcess();
        currentModel    = &currentEvent->GetModel();
    }

// ---------------------------------------------------------------------------

void Scheduler::Terminate (DynamicalObject& d)
    {
        dynObjToDelete->Insert (&d);
    }

// ---------------------------------------------------------------------------

void Scheduler::PrepareDeletionOf (Model& m)
    {
        if (&currentEvent->GetModel() == &m)
            currentEvent = &ModelComponent::NullEvent();
        if (&currentEntity->GetModel() == &m)
            currentEntity   =
            currentProcess  = &ModelComponent::NullProcess();
    }

// ---------------------------------------------------------------------------

void Scheduler::Debug ()
    {
        if (!currentModel->GetExperiment().DebugIsOn())
            return;

        strstream ss;
        long      oflgs = ss.flags(ios::showpoint | ios::fixed | ios::right);
        int       tw = currentModel->GetExperiment().TimeWidth();
        int       tp = currentModel->GetExperiment().TimePrecision();
        int       nw = currentModel->GetExperiment().NameWidth();
        int       pw = 8;
        int       qOffset = tw + 3*nw + pw + 5;
```

```
        // Ueberschrift
        ss  << "EventList at ClockTime : " << simClock.Time() << endl
            << resetiosflags(ios::left) << setfill('=')
            << setw (qOffset + nw) << "" << setfill (' ') << endl
            << setiosflags(ios::left) << setw (nw) << "Model" << ' '
            << resetiosflags(ios::left) << setw (tw) << "Time" << ' '
            << setiosflags(ios::left) << setw (nw) << "Event" << ' '
            << setiosflags(ios::left) << setw (nw) << "Entity" << ' '
            << resetiosflags(ios::left) << setw (pw) << "Priority" << ' '
            << setiosflags(ios::left) << setw (nw) << "in Queue" << endl
            << resetiosflags(ios::left) << setfill('=')
            << setw (qOffset + nw) << "" << setfill (' ') << endl
            << resetiosflags(ios::left);
        // current
        EventNote* note = eventList->NewEventNote
                            (simClock.Time(), currentEvent, currentEntity);
        Debug (note, ss, tw, nw, pw, qOffset);
        ss  << resetiosflags(ios::left) << setfill('-')
            << setw (qOffset + nw) << "" << setfill (' ') << endl;
        delete note;
        // der Rest
        note = eventList->FirstEventNote();
        while (note)
        {
            Debug (note, ss, tw, nw, pw, qOffset);
            note = eventList->NextEventNote (note);
        }

        ss  << ends;
        ExperimentManager::Instance().Note (DebugMessage(String(ss)),
                                            *currentModel);
    }

// ---------------------------------------------------------------------------

void Scheduler::Debug (EventNote* note, ostream& os,
                       int tw, int nw, int pw, int qo)
    {   // eine Ereignisnotiz ausgeben
        if (note->event->IsNullEvent() && note->entity->IsNullEntity())
            os << setiosflags(ios::left) << setw(nw)
               << currentModel->Name().Left(nw);
        else
            os << setiosflags(ios::left) << setw(nw)
               << note->model->Name().Left(nw);

        os  << resetiosflags(ios::left) << setw(tw+1)
            << note->time << ' '
            << setiosflags(ios::left) << setw(nw)
            << note->event->Name().Left(nw) << ' '
            << setw(nw) << note->entity->Name().Left(nw)
            << resetiosflags(ios::left) << setw(pw+1);
        if (note->entity->IsNullEntity())
            os << "" << ' ';
        else
            os << note->entity->GetPriority() << ' ';
        os << setiosflags(ios::left);
        // Queues
        QueueLink*  qlink = QueueLink::GetQueueLink (*note->entity);
        while (qlink)
        {
            os << setw(nw) << qlink->GetQueue().Name().Left(nw) << ' ';
            qlink = qlink->Same();
            if (qlink)
                os << endl << setw(qo) << "";
        }
        os  << resetiosflags(ios::left) << endl;
    }

// ---------------------------------------------------------------------------

void Scheduler::preemptCurrent (EventNote* note)
    {
        assert (currentEvent && currentEntity && currentProcess);
        assert (!currentProcess->IsNullProcess());
            // Verdraengung nur wenn Prozess aktiv

        note->time       = simClock.Time();
        EventNote* note2 = eventList->NewEventNote (simClock.Time(),
                                                    currentEvent,
                                                    currentEntity);

        eventList->InsertAsFirst (note2);   // current
        eventList->InsertAsFirst (note);    // Verdraenger
```

```
        // Kontrolle ans Hauptprogramm (->ProcessNextEventNote())
        Coroutine::MainCoroutine()->Transfer();
    }

// ---------------------------------------------------------------------------
```

simclock.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          simclock.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#ifndef SIMCLOCK_H
#define SIMCLOCK_H

// ---------------------------------------------------------------------------

#include "observab.h"

#include "simtime.h"
#include "boolean.h"

// ---------------------------------------------------------------------------

class SimClock : public Observable
    /*  kapselt die aktuelle Simulationszeit und sorgt dafuer, dass
        sehr dicht beienanderliegende Zeitpunkte keine Zeitfortschreibng
        bewirken. Ausserdem werden bei Fortschreibung automatisch alle
        registrierten Observer benachrichtigt (Observable).
    */
{
    public:
                    SimClock (SimTime epsilon);
        virtual    ~SimClock ();

                SimTime Epsilon() const;   // liefert die groesste Zeitspanne,
                                           // die nicht zur Fortschreibung
                                           // der Uhr fuehrt
                SimTime Time () const;

                SimTime SetTime (SimTime t);
                                    // Setzt die Zeit, nachdem alle Observer
                                    // benachrichtigt wurden
                                    // und gibt die neue Zeit zurueck
    private:
                SimTime epsilon;
                SimTime time;
                bool    locked;     // true waehrend SetTime()
};

// ---------------------------------------------------------------------------

#endif  //  SIMCLOCK_H
```

simclock.cc

```
// ---------------------------------------------------------------------------
```

```
//
// Datei
//          simclock.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#include "simclock.h"
#include "observer.h"
#include "observab.h"
#include "simtime.h"
#include <assert.h>

// ---------------------------------------------------------------------------

SimClock::SimClock (SimTime epsi)
    :   epsilon (epsi),
        time    (0.0),
        locked  (false)
    {}

// ---------------------------------------------------------------------------

SimClock::~SimClock ()
    {}

// ---------------------------------------------------------------------------

SimTime SimClock::Epsilon () const
    {   return epsilon; };

// ---------------------------------------------------------------------------

SimTime SimClock::Time () const
    {   return time; };

// ---------------------------------------------------------------------------

SimTime SimClock::SetTime (SimTime t)
    {
        assert(!locked);
        if (t > (time + epsilon))
        {
            locked = true;
            NotifyObservers();
            time = t;
            locked = false;
        }
        return time;
    }

// ---------------------------------------------------------------------------
```

simtime.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          simtime.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
```

```
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------
//
// Beschreibung
//
//          Die Klasse SimTime repraesentiert den Datentyp fuer die
//          Zeitdarstellung in DESMO-C. Sie entspricht dem in DESMO an
//          verschiedenen Stellen definierten Typ SimTime.
//          Intern wird die Zeit als Fliesskommazahl vom Typ double
//          gespeichert.
//
// ---------------------------------------------------------------------------
//
// Weiterentwicklung von:
//
// Diplomarbeit
//
//          Entwurf und Realisierung eines objektorientierten
//          Simulationspakets in C++
//
// Author
//          Heiko Weber
//
// Beschreibung
//
//          Die Klasse SimTime stellt den Datentyp fuer die
//          Simulationsuhr zur Verfuegung.
//
// ---------------------------------------------------------------------------

#ifndef  SIMTIME_H
#define  SIMTIME_H

// ---------------------------------------------------------------------------

#include <iostream.h>                   // fuer Ein-/Ausgabe
#include "str.h"

// ---------------------------------------------------------------------------

class   SimTime
{
    public:
        enum eOutputMode      { Floating,
                                Digital };
        enum eUnits           { Hours,      // 1.0 == 1 Hour
                                Minutes,    // 1.0 == 1 Minutes
                                Seconds };  // 1.0 == 1 Second

        // Konstruktoren
                                SimTime(void) : time(0.0) {}
                                SimTime(double t) : time(t) {}
        // Destructor
        virtual                ~SimTime(void) {}

        // Query
        double                  Time(void) const { return time; }
        friend ostream&         operator<<(ostream&, const SimTime&);
        friend istream&         operator>>(istream&, SimTime&);
        static unsigned         Width(void) { return width; }

                String          AsString(int width, int precision) const;

        // Operators
        SimTime&         operator=(const SimTime& t)
                                { time = t.time; return *this; }
        SimTime&         operator+=(const SimTime& t)
                                { time += t.time; return *this; }
        SimTime&         operator-=(const SimTime& t)
                                { time -= t.time; return *this; }
        SimTime&         operator*=(const SimTime& t)
                                { time *= t.time; return *this; }
        SimTime&         operator/=(const SimTime& t)
                                { time /= t.time; return *this; }
        SimTime          operator/(const SimTime& t) const
                                { return SimTime(time / t.time); }
        SimTime          operator*(const SimTime& t) const
                                { return SimTime(time * t.time); }
        SimTime          operator+(const SimTime& t) const
                                { return SimTime(time + t.time); }
```

```
        SimTime          operator-(const SimTime& t) const
                                { return SimTime(time - t.time); }
        int              operator<(const SimTime& t) const
                                { return time < t.time; }
        int              operator>(const SimTime& t) const
                                { return time > t.time; }
        int              operator<=(const SimTime& t) const
                                { return time <= t.time; }
        int              operator>=(const SimTime& t) const
                                { return time >= t.time; }
        int              operator==(const SimTime& t) const
                                { return time == t.time; }
        int              operator!=(const SimTime& t) const
                                { return time != t.time; }

        static SimTime      Now();
        static void         SetFloating();
        static void         SetDigital(eUnits = Hours);

    protected:
        static unsigned     width;
        static eOutputMode  mode;
        static eUnits       units;

    private:
        double              time;       // current time value

};

// ---------------------------------------------------------------------------

#endif  // SIMTIME_H
```

simtime.cc

```
// ---------------------------------------------------------------------------
//
// Datei
//          simtime.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------
//
// Weiterentwicklung von:
//
// Diplomarbeit
//
//          Entwurf und Realisierung eines objektorientierten
//          Simulationspakets in C++
//
// Author
//          Heiko Weber
//
// Beschreibung
//
//          Die Klasse SimTime stellt den Datentyp fuer die
//          Simulationsuhr zur Verfuegung.
//
// ---------------------------------------------------------------------------

#include <iomanip.h>
#include <math.h>
#include "simtime.h"
#include "str.h"
#include "strstr.h" // (TS) fuer strstream (jetzt: ostrstream)

// ---------------------------------------------------------------------------
```

```
unsigned             SimTime::width = 10;
SimTime::eUnits      SimTime::units = SimTime::Hours;
SimTime::eOutputMode SimTime::mode  = SimTime::Floating;

// ---------------------------------------------------------------------------

const SimTime   cNow    = -12345;

// ---------------------------------------------------------------------------
// friend of SimTime

ostream& operator<<(ostream &out, const SimTime& st)
{
    strstream   ss;

    if (st.mode == SimTime::Floating) {
        ss.flags(ios::showpoint | ios::fixed | ios::right); // (TS)
        ss.precision(3);
        ss << setiosflags(ios::showpoint | ios::fixed | ios::right)
           << st.time;
    } else {
        double d = 0, h, m, s, t;

        switch(st.units) {
            case SimTime::Hours:
                h = floor(st.time); t  = (st.time - h) * 60.0;
                m = floor(t);       t -= m;
                s = floor(t * 60.0);
                break;
            case SimTime::Minutes:
                h = floor(st.time / 60.0); t  = st.time - (h * 60.0);
                m = floor(t);              t -= m;
                s = floor(t * 60);
                break;
            case SimTime::Seconds:
                h = floor(st.time / 3600.0); t  = st.time - (h * 3600.0);
                m = floor(t / 60.0);         t -= m * 60.0;
                s = floor(t);
                break;
        }
        if (h > 23) {
            d =  floor(h / 24.0);
            h -= (d * 24.0);
        }
        if (d < 10) {
            if (d > 0.9)
                ss << d << "/" << setw(2);
            ss << h << ':';
            ss.fill('0');
            ss << setw(2) << m << ':' << setw(2) << s;
        } else {
            ss << d << "/" << setw(2) << h << ':';
            ss.fill('0');
            ss << setw(2) << m;
        }
    }
    ss << ends;
    out << String(ss);
    return out;
}

// ---------------------------------------------------------------------------

istream& operator>>(istream &in, SimTime& st)
{
    in >> st.time;
    return in;
}

// ---------------------------------------------------------------------------

SimTime SimTime::Now()
{
    return cNow;
}

// ---------------------------------------------------------------------------
// (TS)
String SimTime::AsString (int w, int p) const
{
    strstream   ss;

    if (w <= 0) return "";
```

```
    if (p <= 0) p = 0;

    ss.flags(ios::showpoint | ios::fixed | ios::right);
    ss.precision(p);
    ss << time << ends;

    String      s(ss);

    if (s.Length() <= w) return s;
    return s.Left (w);
}

// ---------------------------------------------------------------------------

void SimTime::SetDigital(eUnits u)
{
    mode  = Digital;
    units = u;
}

// ---------------------------------------------------------------------------

void SimTime::SetFloating(void)
{
    mode  = Floating;
}

// ---------------------------------------------------------------------------
```

statobj.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          statobj.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#ifndef STATISTICOBJECT_H
#define STATISTICOBJECT_H

// ---------------------------------------------------------------------------

#include "reportab.h"   // Basisklasse
#include "observer.h"   // Basisklasse

#include "str.h"

// ---------------------------------------------------------------------------

class StatisticObject : public Reportable, public Observer
{
    public:
                    StatisticObject (           Model& owner,
                                        const   String& name = "",
                                        bool    showInReport = true,
                                        bool    showInTrace  = false);

        virtual void            Update () = 0;
        virtual void            NoteChange (Observable*);   // von Observer

                String          ClassName () const;
    protected:
                void            traceUpdate ();
    private:
```

```
};

// ---------------------------------------------------------------------------

#endif  // STATISTICOBJECT_H
```

statobj.cc

```
// ---------------------------------------------------------------------------
//
// Datei
//          statobj.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#include "statobj.h"

// ---------------------------------------------------------------------------

static const char* className = "StatisticObject";

// ---------------------------------------------------------------------------

StatisticObject::StatisticObject (          Model& owner,
                                    const   String& name,
                                    bool    showInReport,
                                    bool    showInTrace)
    :   Reportable(owner, name, showInReport, showInTrace),
        Observer()
    {}

// ---------------------------------------------------------------------------

void StatisticObject::NoteChange (Observable*)
    {
        Update();
    }

// ---------------------------------------------------------------------------

String StatisticObject::ClassName () const
    {
        return className;
    }

// ---------------------------------------------------------------------------

void StatisticObject::traceUpdate ()
    {
        if (TraceIsOn())
            TraceNote ("updates " + QuotedName());
    }

// ---------------------------------------------------------------------------
```

stdoutp.h

```
// ---------------------------------------------------------------------------
//
// Datei
```

```
//          stdoutp.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ------------------------------------------------------------------------------

#ifndef STDOUTPUT_H
#define STDOUTPUT_H

// ------------------------------------------------------------------------------

#include "messager.h"   // Basisklasse

#include "simtime.h"
#include "str.h"

// ------------------------------------------------------------------------------

class Output : public MessageReceiver
{
    public:
                    Output (ostream& os, unsigned width = 100);
                    Output (const   String&     fileName,
                            const   String&     extension,  // inkl '.'
                                    unsigned    width = 100);
        virtual    ~Output ();

                void        Rename (const String& name);
                unsigned    GetWidth() const;
                bool        Empty() const;
    protected:
                ostream&    GetOstream();
    private:
        ostream&    os;
        bool        isFile;     // true => os ist ein fstream
        String      fileName;
        String      ext;
        unsigned    width;
};

// ------------------------------------------------------------------------------
// ------------------------------------------------------------------------------

class StdOutput : public Output
{
    public:
                    StdOutput (ostream& os, unsigned width = 78);
                    StdOutput ( const   String&     fileName,
                                const   String&     extension,  // inkl '.'
                                        unsigned    width = 78);

        virtual String  GetOutputTitle() const = 0;

        virtual void    SwitchOn  ( const   Message&    msg);
        virtual void    SwitchOff ( const   Message&    msg,
                                    const   String&     what);
    protected:
                void    Box  (const String& title);
                void    Box2 (const String& title1, const String& title2);
                void    Line (char c = '-');
                void    ClockTime (const SimTime&);
        virtual void    WriteHeader ();

                void    wrap (const String&     s,
                                    unsigned    offset,
                                    unsigned    indent = 0);
};

// ------------------------------------------------------------------------------
// ------------------------------------------------------------------------------

class StdDebug : public StdOutput
{
    public:
```

```
                    StdDebug (ostream& os, unsigned width = 78);
                    StdDebug (  const   String&     fileName,
                                const   String&     extension = ".dbg",
                                        unsigned    width = 78);

        virtual String  GetOutputTitle() const;
        virtual void    Note (const Message&);
};

// ------------------------------------------------------------------------------

class StdError : public StdOutput
{
    public:
                    StdError (ostream& os, unsigned width = 78);
                    StdError (  const   String&     fileName,
                                const   String&     extension = ".err",
                                        unsigned    width = 78);

        virtual String  GetOutputTitle() const;
        virtual void    Note (const Message&);
};

// ------------------------------------------------------------------------------

class StdGlError : public StdError
{
    public:
                    StdGlError (ostream& os, unsigned width = 78);
                    StdGlError (    const   String&     fileName,
                                    const   String&     extension = ".glb",
                                            unsigned    width = 78);

        virtual String  GetOutputTitle() const;
};

// ------------------------------------------------------------------------------

class StdReport : public StdOutput
{
    public:
                    StdReport (ostream& os, unsigned width = 79);
                    StdReport ( const   String&     fileName,
                                const   String&     extension = ".rpt",
                                        unsigned    width = 79);

        virtual String  GetOutputTitle() const;
        virtual void    Note (const Message&);
        virtual void    TakeReporter (Reporter&);
    protected:
                void    WriteBeginOfGroup (Reporter&);
    private:
                Reporter*   lastReporter;
};

// ------------------------------------------------------------------------------

class Event;
class Entity;
class Model;

class StdTrace : public StdOutput
{
    public:
                    StdTrace (ostream& os, unsigned width = 90);
                    StdTrace (  const   String&     fileName,
                                const   String&     extension = ".trc",
                                        unsigned    width = 90);

        virtual String  GetOutputTitle() const;
        virtual void    WriteHeader ();
        virtual void    Note (const Message&);
    private:
                void    writeMessage (const Message&);

        const   Event*  lastEvent;
        const   Entity* lastEntity;
        const   Model*  lastModel;
                SimTime lastTime;
};

// ------------------------------------------------------------------------------

#endif  // STDOUTPUT_H
```

stdoutp.cc

```
// --------------------------------------------------------------------------
//
// Datei
//          stdoutp.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// --------------------------------------------------------------------------

#include "stdoutp.h"

#include "event.h"
#include "entity.h"
#include "experime.h"
#include "experimm.h"
#include "model.h"
#include "msgtypes.h"
#include "portable.h"   // fuer tell und seek
#include "reporter.h"
#include <fstream.h>
#include <iostream.h>
#include <iomanip.h>
#include <stdio.h>

// --------------------------------------------------------------------------

Output::Output (ostream& OS, unsigned w)
    :   MessageReceiver (),
        os              (OS),
        isFile          (false),
        fileName        (""),
        ext             (""),
        width           (w)
    {}

// --------------------------------------------------------------------------

Output::Output (const String& fName, const String& extension, unsigned w)
    :   MessageReceiver (),
        os              (*new ofstream ((fName + extension).Get())),
        isFile          (true),
        fileName        (fName),
        ext             (extension),
        width           (w)
    {
        GetOstream() << setiosflags(ios::right);
    }

// --------------------------------------------------------------------------

Output::~Output ()
    {
        if (isFile)
        {
            if (Empty())
                Rename("");
            delete& os;
            isFile = false;
        }
    }

// --------------------------------------------------------------------------

unsigned Output::GetWidth () const
    {
        return width;
    }

// --------------------------------------------------------------------------
```

```
bool Output::Empty () const
    {
        return os.tellp() == 0;
    }

// ------------------------------------------------------------------------------

ostream& Output::GetOstream ()
    {
        return os;
    }

// ------------------------------------------------------------------------------

void Output::Rename (const String& s)
    {
        if (isFile && s != fileName)
        {
            ofstream& file = (ofstream&)os;
            streampos pos = file.tellp();

            file.close();
            if (pos == 0)
                remove ((fileName + ext).Get());
            fileName = s;
            if (fileName != "")
                file.open ((fileName + ext).Get());
        }
//      else
//          file.seekp(0);
    }

// ------------------------------------------------------------------------------
// ------------------------------------------------------------------------------

StdOutput::StdOutput (ostream& os, unsigned w)
    :   Output  (os, w)
    {}

// ------------------------------------------------------------------------------

StdOutput::StdOutput (const String& name, const String& ext, unsigned w)
    :   Output  (name, ext, w)
    {}

// ------------------------------------------------------------------------------

void StdOutput::Box (const String& s)
{
    ostream& os = GetOstream();
    char star  = '*';
    char c     = os.fill(star);
    int  w     = GetWidth();
    int  l1    = w - 2 - s.Length();

    os << setw(w) << "" << endl;
    os.fill(c);
    os  << star << setw(w-2) << "" << star << endl
        << star << setw(l1/2) << "" << s
        << setw(l1 - (l1/2)) << "" << star << endl
        << star << setw(w-2) << "" << star << endl;
    os.fill(star);
    os  << setw(w) << "" << endl;
    os.fill(c);
}

// ------------------------------------------------------------------------------

void StdOutput::Box2 (const String& s1, const String& s2)
{
    ostream& os= GetOstream();
    char star  = '*';
    char c     = os.fill(star);
    int  w     = GetWidth();
    int  l1    = w - 2 - s1.Length(),
         l2    = w - 2 - s2.Length();

    os  << setw(w) << "" << endl;
    os.fill(c);
    os  << star << setw(w-2) << "" << star << endl
        << star << setw(l1/2) << "" << s1
        << setw(l1 - (l1/2)) << "" << star << endl
        << star << setw(w-2) << "" << star << endl
        << star << setw(l2/2) << "" << s2
```

```
        << setw(l2 - (l2/2)) << "" << star << endl
        << star << setw(w-2) << "" << star << endl;
    os.fill(star);
    os  << setw(w) << "" << endl;
    os.fill(c);
}

// ---------------------------------------------------------------------------

void StdOutput::Line (char c)
{
    ostream&    os      = GetOstream();
    char        ofill   = os.fill(c);
    int         w       = GetWidth();

    os  << setw (w) << "" << endl;
    os.fill(ofill);
}

// ---------------------------------------------------------------------------

void StdOutput::ClockTime (const SimTime& t)
{
    ostream& os = GetOstream();
    strstream   ss;

    ss << "Clock Time = ";
    ss << t << ends;

    String s(ss);
    int  w = GetWidth();
    int  l = w - s.Length();

    os << setw(l/2) << "" << s << endl;
}

// ---------------------------------------------------------------------------

void StdOutput::SwitchOn (const Message& msg)
    {
        ostream& os = GetOstream();

        ClockTime (msg.Time());
        Box2 (  String("Experiment: ") + msg.GetExperiment().Name(),
                GetOutputTitle());
        GetOstream() << endl;
        WriteHeader();
    }

// ---------------------------------------------------------------------------

void StdOutput::SwitchOff ( const   Message&    msg,
                            const   String&     what)
    {
        ostream& os = GetOstream();

        os << endl;
        ClockTime (msg.Time());
        Box (what + " switched off");
        os << endl;
    }

// ---------------------------------------------------------------------------

void StdOutput::WriteHeader ()
    {}

// ---------------------------------------------------------------------------

void StdOutput::wrap (const String& s, unsigned offset, unsigned indent)
    {
        ostream&    os      = GetOstream();
        const       w1      = GetWidth() - offset;
        const       w2      = w1 - indent;
        int         w       = w1;
        int         len     = s.Length();
        int         i       = 0;

        if (len <= w)
            os << s << endl;
        else
        {
            String rest = s;
            while (len > w && len > 0)
```

```
            {
                i = 0;
                int j = i = rest.Find (' ', i);
                while (j <= w && j >= 0)
                {
                    i = j;
                    j = rest.Find (' ', i+1);
                }
                if (len > w && i > 0)
                {   // Es wird noch eine Zeile benoetigt
                    os  << rest.Left (i) << endl
                        << setiosflags(ios::left)
                        << setw(offset + indent) << "";
                    w = w2;
                }
                len -= (i + 1);
                rest = rest.Right (len);
            }
            os  << rest << endl;
        }
    }

// ------------------------------------------------------------------------------
// ------------------------------------------------------------------------------

StdDebug::StdDebug (ostream& os, unsigned w)
    :   StdOutput   (os, w)
    {}

// ------------------------------------------------------------------------------

StdDebug::StdDebug (const String& name, const String& ext, unsigned w)
    :   StdOutput   (name, ext, w)
    {}

// ------------------------------------------------------------------------------

String StdDebug::GetOutputTitle () const
    {
        return "Debug";
    }

// ------------------------------------------------------------------------------

void StdDebug::Note (const Message& msg)
    {
        const char separator = '*';

        if (msg.Type() == Message::error)
        {
            GetOstream()    << endl
                            << msg.CodeText() << ": " << msg.Description()
                            << endl << endl;
            Line (separator);
            return;
        }

        if (msg.Type() == Message::trace && msg.Code() == Message::normal)
        {
            GetOstream()<< endl << msg.Time() << ": ";

            if (!msg.GetEvent().IsNullEvent())
                GetOstream() << msg.GetEvent().QuotedName() << " of ";

            if (!msg.GetEntity().IsNullEntity())
                    GetOstream() << msg.GetEntity().QuotedName() << " of ";

            GetOstream()<< "Model " << msg.GetModel().QuotedName() << ' '
                        << msg.Description() << endl << endl;
            Line (separator);
            return;
        }

        if (msg.Type() == Message::debug)
        {
            switch (msg.Code())
            {
                case Message::normal:
                    GetOstream()<< endl << msg.Description()
                                << endl;
                    Line (separator);
                    break;
                case Message::switchOn:
                    SwitchOn (msg);
```

```
                    break;
                case Message::switchOff:
                    SwitchOff (msg, "debugging");
                    break;
                default:
                    ;
            }
        }
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

StdError::StdError (ostream& os, unsigned w)
    :   StdOutput   (os, w)
    {}

// ---------------------------------------------------------------------------

StdError::StdError (const String& name, const String& ext, unsigned w)
    :   StdOutput   (name, ext, w)
    {}

// ---------------------------------------------------------------------------

String StdError::GetOutputTitle () const
    {
        return "Error";
    }

// ---------------------------------------------------------------------------

void StdError::Note (const Message& msg)
    {
        const cIndent = 18;
        String what     = msg.Consequences ();

        if (Empty())
            SwitchOn (msg);
        GetOstream()
            << "Current Time    : " << msg.Time() << endl
            << "Kind of Error   : " << msg.CodeText() << endl
            << "Description     : "; wrap (msg.Description(), cIndent);
        GetOstream()
            << "Location        : " << msg.Location () << endl
            << "Current Model   : " << msg.GetModel().Name() << endl;
        if (!msg.GetEvent().IsNullEvent())
            GetOstream()
            << "Current Event   : " << msg.GetEvent().Name() << endl;

        if (!msg.GetEntity().IsNullEntity() && msg.GetEntity().IsProcess())
            GetOstream()
            << "Current Process : " << msg.GetEntity().Name() << endl;
        else
            GetOstream()
            << "Current Entity  : " << msg.GetEntity().Name() << endl;

        if (what.Length() > 0)
            GetOstream()
                << "Consequences    : "; wrap (what, cIndent);

        what = msg.Hint();
        if (what.Length() > 0)
            GetOstream()
                << "Hint            : "; wrap (what, cIndent);

        GetOstream()
            << endl;
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

StdGlError::StdGlError (ostream& os, unsigned w)
    :   StdError    (os, w)
    {}

// ---------------------------------------------------------------------------

StdGlError::StdGlError (const String& name, const String& ext, unsigned w)
    :   StdError    (name, ext, w)
    {}

// ---------------------------------------------------------------------------
```

```
String StdGlError::GetOutputTitle () const
    {
        return "Global Errors";
    }

// --------------------------------------------------------------------------
// --------------------------------------------------------------------------

StdReport::StdReport (ostream& os, unsigned w)
    :   StdOutput   (os, w),
        lastReporter(0)
    {}

// --------------------------------------------------------------------------

StdReport::StdReport (const String& name, const String& ext, unsigned w)
    :   StdOutput   (name, ext, w),
        lastReporter(0)
    {}

// --------------------------------------------------------------------------

String StdReport::GetOutputTitle () const
    {
        return "Report";
    }

// --------------------------------------------------------------------------

void StdReport::Note (const Message& msg)
    {
        if (Empty())
            SwitchOn (msg);
        ostream&    os = GetOstream();
        switch (msg.Code())
        {
            case Message::descriptionAsBox:
                os << endl;
                ClockTime(msg.Time());
                Box (msg.Description());
                os << endl;
                break;
            default:
                os << msg.Description() << endl;
        }
    }

// --------------------------------------------------------------------------

void StdReport::WriteBeginOfGroup (Reporter& r)
    {
        ostream&    os = GetOstream();
        int         rw = GetWidth();

        if (r.HasTitle())
        {
            os << endl;
            r.WriteTitle (os, rw);
            r.UnderscoreTitle (os,  rw) << endl;
        }
        if (r.HasHeader())
        {
            r.WriteHeader (os) << endl;
            r.UnderscoreHeader (os) << endl;
        }
    }

// --------------------------------------------------------------------------

void StdReport::TakeReporter (Reporter& r)
    {
        if (Empty())
            SwitchOn (ReportMessage (""));  // Dummy-Message
        ostream&    os = GetOstream();
        int         rw = GetWidth();

        if (!lastReporter)
            ;   // WriteBeginOfGroup (rf, r);
        else if (lastReporter->GetGroupID() != r.GetGroupID())
            WriteBeginOfGroup (r);

        lastReporter = &r;
```

```
        r.WriteAsLine (os, rw) << endl;
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

StdTrace::StdTrace (ostream& os, unsigned w)
    :   StdOutput   (os, w),
        lastEvent   (0),
        lastEntity  (0),
        lastModel   (0),
        lastTime    (-2.0)
    {}

// ---------------------------------------------------------------------------

StdTrace::StdTrace (const String& name, const String& ext, unsigned w)
    :   StdOutput   (name, ext, w),
        lastEvent   (0),
        lastEntity  (0),
        lastModel   (0),
        lastTime    (-2.0)
    {}

// ---------------------------------------------------------------------------

String StdTrace::GetOutputTitle () const
    {
        return "Trace";
    }

// ---------------------------------------------------------------------------

void StdTrace::WriteHeader ()
    {
        unsigned    nw = ExperimentManager::Instance().
                            CurrentExperiment().NameWidth();
        unsigned    tw = ExperimentManager::Instance().
                            CurrentExperiment().TimeWidth();

        ostream& os = GetOstream();
        os  << endl << setw (nw) << setiosflags(ios::left) << "Model" << ' '
            << resetiosflags(ios::left) << setw (tw) << "Time" << ' '
            << setw (nw) << setiosflags(ios::left) << "Event" << ' '
            << setw (nw) << setiosflags(ios::left) << "Entity" << ' '
            << "Action(s)" << endl << resetiosflags(ios::left);
        Line();
    }

// ---------------------------------------------------------------------------

void StdTrace::Note (const Message& msg)
    {
        if (msg.Type() == Message::error)
            writeMessage (msg);

        if (msg.Type() == Message::trace)
        {
            switch (msg.Code())
            {
                case Message::normal:
                    writeMessage (msg);
                    break;
                case Message::switchOn:
                    SwitchOn (msg);
                    break;
                case Message::switchOff:
                    SwitchOff (msg, "tracing");
                    break;
                default:
                    ;
            }
        }
    }

// ---------------------------------------------------------------------------

void StdTrace::writeMessage (const Message& msg)
    {
        unsigned    nw = ExperimentManager::Instance().
                            CurrentExperiment().NameWidth();
        unsigned    tw = ExperimentManager::Instance().
                            CurrentExperiment().TimeWidth();
        unsigned    descrOffset = tw + 3 * nw + 4;
```

```
        unsigned    indent      = 2;
        ostream&    os = GetOstream();

        // Model
        if (lastModel == &msg.GetModel())
            os << setiosflags(ios::left) << setw (nw+1) << "";
        else
            os << setiosflags(ios::left) << setw (nw)
               << (lastModel = &msg.GetModel())->Name().Left(nw) << ' ';

        // SimTime
        if (lastTime == msg.Time())
            os << resetiosflags(ios::left) << setw (tw+1) <<  "";
        else
            os << resetiosflags(ios::left) << setw (tw)
               << (lastTime = msg.Time()) << ' ';

        {
            bool skipEntity = false;

            // Event
            if (lastEvent == &msg.GetEvent())
                os << setiosflags(ios::left) << setw (nw+1) << "";
            else
            {
                os << setiosflags(ios::left) << setw (nw)
                   << msg.GetEvent().Name().Left(nw) << ' ';
                if (msg.GetEvent().IsExternal())
                {
                    os << setiosflags(ios::left) << setw (nw)
                       << "External" << ' ';
                    lastEntity = &msg.GetEntity();
                    skipEntity = true;
                }
                lastEvent = &msg.GetEvent();
            }

            // Entity bzw. Process
            if (!skipEntity)
            {
                if (lastEntity == &msg.GetEntity())
                    os << setiosflags(ios::left) << setw (nw+1) << "";
                else
                {
                    os << setiosflags(ios::left) << setw (nw)
                       << msg.GetEntity().Name().Left(nw) << ' ';
                    lastEntity = &msg.GetEntity();
                }
            }
        }

        // Description
        if (msg.Type() == Message::error)
            switch (msg.Code())
            {
                case Message::fatalError:
                case Message::normalError:
                case Message::warning:
                    os << msg.CodeText() << ": ";
                    descrOffset += msg.CodeText().Length() + 2;
                    indent = 0;
                    break;
                default:
                    ;
            }
        wrap (msg.Description(), descrOffset, indent);
        os << resetiosflags(ios::left);
    }

// ---------------------------------------------------------------------------
```

str.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          str.h
//
```

```
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------
//
// Beschreibung
//
//          Die Klasse String stellt verschieden Funktionen zur
//          Verarbeitung von Zeichenketten zur Verfuegung.
//
// ---------------------------------------------------------------------------
//
// Weiterentwicklung von:
//
// Diplomarbeit
//
//          Entwurf und Realisierung eines objektorientierten
//          Simulationspakets in C++
//
// Author
//          Heiko Weber
//
// Beschreibung
//
//          Die Klasse String stellt String-Funktionen bereit.
//
// ---------------------------------------------------------------------------

#ifndef  STR_H
#define  STR_H

// ---------------------------------------------------------------------------

#include "boolean.h"
#include "strstr.h"

// ---------------------------------------------------------------------------

class String {
private:
    unsigned                len, buflen;
    char                   *buf;

public:
    // Konstruktoren
            String(void);
            String(const char*);
            String(const String&);
            String(const String&, unsigned);
            String(char);
            String(char, unsigned);
            String(strstream&);
            String(const void*);
            String(int);
            String(unsigned);
            String(long);
            String(unsigned long);
            String(double);
            String(bool);

    // Destruktor
    virtual ~String(void) { delete[] buf; }

    // Selektoren
            const char*     Get(void) const { return buf; }
            unsigned        Length(void) const { return len; }
        // Vergleichsoperatoren
            bool            operator==(const String &s2) const;
    friend  bool            operator==(const char* s1, const String &s2);
            bool            operator!=(const String &s2) const;
    friend  bool            operator!=(const char* s1, const String &s2);
            bool            operator<(const String &s2) const;
    friend  bool            operator<(const char* s1, const String &s2);
            bool            operator<=(const String &s2) const;
    friend  bool            operator<=(const char* s1, const String &s2);
```

```
            bool            operator>(const String &s2) const;
    friend  bool            operator>(const char* s1, const String &s2);
            bool            operator>=(const String &s2) const;
    friend  bool            operator>=(const char* s1, const String &s2);
            int             Compare(const String&) const;
            int             Compare(const char*) const;
        // Suche
            int             Find(const char, unsigned pos = 0) const;
            int             Find(const String&, unsigned pos = 0) const;
        // Teilstrings
            String          Left (int) const;
            String          Right(int) const;
            String          Substr(int from, int n = -1) const;
        // Ausgabe
    friend  ostream&        operator<<(ostream &o, const String &s);

    // Manipulatoren
        // Zuweisung
            String&         operator=(const String&);
            String&         operator=(const char*);
            String&         operator+=(const String&);
        // Konkatenation
            String          operator+(const String&) const;
    friend  String          operator+(const char*, const String&);
        // Sonstige
            int             Replace(const String &substr,
                                    const String &repstr, unsigned pos = 0);
            int             Replace(const char substr,
                                    const char repstr, unsigned pos = 0);
            String&         Insert(const String&, unsigned pos = 0);
            String&         LTrim(void);
            String&         RTrim(void);
            String&         Trim(void) { return RTrim().LTrim(); }
};

// ------------------------------------------------------------------------------

#endif  // STR_H
```

str.cc

```
// ------------------------------------------------------------------------------
//
// Datei
//          str.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ------------------------------------------------------------------------------
//
// Weiterentwicklung von:
//
// Diplomarbeit
//
//          Entwurf und Realisierung eines objektorientierten
//          Simulationspakets in C++
//
// Author
//          Heiko Weber
//
// Beschreibung
//
//          Die Klasse String stellt String-Funktionen bereit.
//
// ------------------------------------------------------------------------------

#include "strstr.h" // fuer strstream
#include <string.h>
```

```
#include <ctype.h>
#include "str.h"

// ---------------------------------------------------------------------------

String::String(void)
    : len(0), buflen(1), buf(new char[1])
{
    *buf = 0;
}

// ---------------------------------------------------------------------------

String::String(const char *s)
    : len(strlen(s))
{
    strcpy(buf = new char[ buflen = len + 1 ], s);
}

// ---------------------------------------------------------------------------

String::String(const String &s)
    : len(s.len), buflen(s.len + 1), buf(new char[s.len + 1])
{
    strcpy(buf, s.buf);
}

// ---------------------------------------------------------------------------

String::String(const String &s, unsigned maxlen)    // TS
    : len(s.len)
{
    if (maxlen < 0)                  // TS
        len = 0;                     // TS
    else                             // TS
        if (len > maxlen) len = maxlen;
    strncpy(buf = new char[ buflen = len + 1 ], s.buf, len);
    buf[ len ] = 0;
}

// ---------------------------------------------------------------------------

String::String(char ch)
    : len(1), buflen(2), buf(new char[2])
{
    buf[0] = ch;
    buf[1] = 0;
}

// ---------------------------------------------------------------------------

String::String(char ch, unsigned cnt)
    : len(cnt), buflen(cnt + 1), buf(new char[cnt + 1])
{
    if (cnt)
        memset(buf, ch, cnt);
    buf[cnt] = 0;
}

// ---------------------------------------------------------------------------

String::String(strstream &s)
    : len(0), buflen(0), buf(0)
{
    if (s.pcount())
    {
        buflen  = s.pcount();
        buf     = new char [buflen];
        strncpy (buf, s.str(), buflen-1);   // str() friert s ein
        s.rdbuf()->freeze(0);               // wieder auftauen
        buf [buflen -1] = 0;
        len = strlen (buf);
    }
    else
    {
        buflen  = 1 + (len = 0);
        buf     = new char [1];
        *buf    = 0;
    }
}

// ---------------------------------------------------------------------------

String::String(const void *p)
```

```
    : len(0), buflen(0), buf(0)
{
    strstream s;
    s << (void*) p << ends;
    buf = s.str();
    len = strlen (buf);
    buflen = len + 1;
}

// ----------------------------------------------------------------------------

String::String(int i)
    : len(0), buflen(0), buf(0)
{
    strstream s;
    s << i << ends;
    buf = s.str();
    len = strlen (buf);
    buflen = len + 1;
}

// ----------------------------------------------------------------------------

String::String(unsigned u)
    : len(0), buflen(0), buf(0)
{
    strstream s;
    s << u << ends;
    buf = s.str();
    len = strlen (buf);
    buflen = len + 1;
}

// ----------------------------------------------------------------------------

String::String(long n)
    : len(0), buflen(0), buf(0)
{
    strstream s;
    s << n << ends;
    buf = s.str();
    len = strlen (buf);
    buflen = len + 1;
}

// ----------------------------------------------------------------------------

String::String(unsigned long n)
    : len(0), buflen(0), buf(0)
{
    strstream s;
    s << n << ends;
    buf = s.str();
    len = strlen (buf);
    buflen = len + 1;
}

// ----------------------------------------------------------------------------

String::String(double d)
    : len(0), buflen(0), buf(0)
{
    strstream s;
    s << d << ends;
    buf = s.str();
    len = strlen (buf);
    buflen = len + 1;
}

// ----------------------------------------------------------------------------

String::String(bool b)
    : len(0), buflen(6), buf(new char[6])
{
    if (b)
        strcpy (buf, "true");
    else
        strcpy (buf, "false");
    len = strlen(buf);
}

// ----------------------------------------------------------------------------

String& String::operator=(const String &rhs)
```

```
{
    if (this != &rhs) {
        if (rhs.len >= buflen) {
            delete[] buf;
            buf = new char[ buflen = rhs.len + 1 ];
        }
        len = rhs.len;
        strcpy(buf, rhs.buf);
    }
    return *this;
}

// ----------------------------------------------------------------------------

String& String::operator=(const char *rhs)
{
    len = strlen(rhs);
    if (len >= buflen) {
        delete[] buf;
        buf = new char[ buflen = len + 1 ];
    }
    strcpy(buf, rhs);
    return *this;
}

// ----------------------------------------------------------------------------

String String::operator+(const String& rhs) const
{
    return String(*this) += rhs;
}

// ----------------------------------------------------------------------------

String operator+(const char* s1, const String& s2)
{
    return String(s1) + s2;
}

// ----------------------------------------------------------------------------

String& String::operator+=(const String &rhs)
{
    if (this != &rhs) {
        len += rhs.len;
        if (len >= buflen) {
            char *tmp = buf;
            buf = new char[ buflen = len + 1];
            strcpy(buf, tmp);
            strcat(buf, rhs.buf);
            delete[] tmp;
        } else
            strcat(buf, rhs.buf);
    } else {
        String tmp = buf;
        *this += tmp;
    }
    return *this;
}

// ----------------------------------------------------------------------------

ostream& operator<<(ostream &o, const String &s)
{
    return o << s.buf;
}

// ----------------------------------------------------------------------------

int String::Compare(const String &s) const
{
    return strcmp(buf, s.buf);
}

// ----------------------------------------------------------------------------

int String::Compare(const char* s) const
{
    return strcmp(buf, s);
}

// ----------------------------------------------------------------------------

String String::Substr(int from, int n) const    // TS
```

```
{
   if (from >= len) return "";
   if (from < 0) from = 0;      // TS
   if (n < 0) n = len;
   return String(&buf[from], n);
}

// ---------------------------------------------------------------------------

int String::Find(const char ch, unsigned pos) const
{
    if (pos < len) {
        const char *s = strchr(buf+pos, ch);
        return s ? int(s - buf) : -1;
    }
    return -1;
}

// ---------------------------------------------------------------------------

int String::Find(const String &str, unsigned pos) const
{
    if (pos < len) {
        const char *s = strstr(buf+pos, str.buf);
        return s ? int(s - buf) : -1;
    }
    return -1;
}

// ---------------------------------------------------------------------------

int String::Replace(const String &sub, const String &rep, unsigned pos)
{
    int p = Find(sub, pos);

    if (p >= 0) {
        String tmp = Substr(0, p) + rep + Substr(p + sub.len);
        *this = tmp;
    }
    return p;
}

// ---------------------------------------------------------------------------

int String::Replace(const char sub, const char rep, unsigned pos)
{
    char *s = (pos < len) ? strchr(buf, sub) : 0;

    if (s != 0) {
        *s = rep;
        return s - buf;
    }
    return -1;
}

// ---------------------------------------------------------------------------

String &String::Insert(const String &s, unsigned pos)
{
   if (pos > len)
       *this += s;
   else if (pos)
       *this = Substr(0, pos) + s + Substr(pos+1);
   else
       *this = s + *this;
   return *this;
}

// ---------------------------------------------------------------------------

String &String::RTrim(void)
{
    char *p = buf;

    while (*p) p++;
    while (p > buf && isspace(p[-1])) p--;
    *p = 0;
    len = unsigned(p - buf);
    return *this;
}

// ---------------------------------------------------------------------------

String &String::LTrim(void)
```

```
{
    char *p = buf, *s = buf;

    while (*p && isspace(*p)) p++;
    len -= unsigned(p - buf);
    while ((*s++ = *p++) != 0)
        {};
    return *this;
}

// ---------------------------------------------------------------------------

bool String::operator==(const String &s2) const
{
    return (len == s2.len) && (Compare(s2) == 0);
}

// ---------------------------------------------------------------------------

bool operator==(const char* s1, const String &s2)
{
    return s2.Compare(s1) == 0;
}

// ---------------------------------------------------------------------------

bool String::operator!=(const String &s2) const
{
    return (len != s2.len) || (Compare(s2) != 0);
}

// ---------------------------------------------------------------------------

bool operator!=(const char* s1, const String &s2)
{
    return s2.Compare(s1) != 0;
}

// ---------------------------------------------------------------------------

bool String::operator<(const String &s2) const
{
    return Compare(s2) < 0;
}

// ---------------------------------------------------------------------------

bool operator<(const char* s1, const String &s2)
{
    return s2.Compare(s1) > 0;
}

// ---------------------------------------------------------------------------

bool String::operator<=(const String &s2) const
{
    return Compare(s2) <= 0;
}

// ---------------------------------------------------------------------------

bool operator<=(const char* s1, const String &s2)
{
    return s2.Compare(s1) >= 0;
}

// ---------------------------------------------------------------------------

bool String::operator>(const String &s2) const
{
    return Compare(s2) > 0;
}

// ---------------------------------------------------------------------------

bool operator>(const char* s1, const String &s2)
{
    return s2.Compare(s1) < 0;
}

// ---------------------------------------------------------------------------

bool String::operator>=(const String &s2) const
{
```

```
    return Compare(s2) >= 0;
}

// --------------------------------------------------------------------------

bool operator>=(const char* s1, const String &s2)
{
    return s2.Compare(s1) <= 0;
}

// --------------------------------------------------------------------------

String String::Left(int length) const
{
    if (length <= 0) return "";
    return String (*this, length);
}

// --------------------------------------------------------------------------

String String::Right(int length) const
{
    if (length <= 0) return "";
    if (length >= len) return *this;
    return Substr (len - length);
}

// --------------------------------------------------------------------------
```

strstr.h

```
// --------------------------------------------------------------------------
//
// Datei
//          strstr.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// --------------------------------------------------------------------------

#ifndef STRSTR_H
#define STRSTR_H

// --------------------------------------------------------------------------

#include <iostream.h>

// --------------------------------------------------------------------------
// --------------------------------------------------------------------------

#ifdef __MWERKS__

#ifdef __MSL__  // Metrowerks Standard Library

#include <sstream.h>
#include <string.h>

// --------------------------------------------------------------------------

class Streambuf : public stringbuf
{
    public:
        void    freeze (int) {};
};

// --------------------------------------------------------------------------

class strstream : public stringstream
```

```
{
    public:
        strstream () : stringstream() {}
        strstream (char* buf, unsigned, ios::openmode how)
            :   stringstream (string (buf), how)
            {}
        unsigned    pcount () { return stringstream::str().length() +1; }
        Streambuf*  rdbuf() { return (Streambuf*)stringstream::rdbuf(); }
        char*       str() { char* buf =
                                new char [stringstream::str().length() + 1];
                            strcpy (buf, stringstream::str().c_str());
                            return buf;
                          }
};

// ---------------------------------------------------------------------------

#else   // __MSL__

#include <strstream.h>
typedef ostrstream strstream;

#endif  // __MSL__
#else   // __MWERKS__

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

#include <strstream.h>

#endif  // __MWERKS__

// ---------------------------------------------------------------------------

#endif  // STRSTR_H
```

sysevent.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          sysevent.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#ifndef SYSTEMEVENT_H
#define SYSTEMEVENT_H

// ---------------------------------------------------------------------------

#include "event.h"      // Basisklasse
#include "observer.h"   // Basisklasse (fuer ReportEvent, ResetEvent)

#include "str.h"

// ---------------------------------------------------------------------------

class SystemEvent : public ExternalEvent
{
    public:
                    SystemEvent (       Model&  owner,
                                const   String& name = "SystemEvent",
                                        bool    showInTrace = false);
};
```

```
// ---------------------------------------------------------------------------

class StopSimEvent : public SystemEvent
{
    public:
                StopSimEvent (Model& owner, bool showInTrace = false);
        virtual void    ExternalEventRoutine ();
};

// ---------------------------------------------------------------------------

class StartTraceEvent : public SystemEvent
{
    public:
                StartTraceEvent (Model& owner, bool showInTrace = false);
        virtual void    ExternalEventRoutine ();
};

// ---------------------------------------------------------------------------

class EndTraceEvent : public SystemEvent
{
    public:
        EndTraceEvent (Model& owner, bool showInTrace = false);
        virtual void    ExternalEventRoutine ();
};

// ---------------------------------------------------------------------------

class StartDebugEvent : public SystemEvent
{
    public:
        StartDebugEvent (Model& owner, bool showInTrace = false);
        virtual void    ExternalEventRoutine ();
};

// ---------------------------------------------------------------------------

class EndDebugEvent : public SystemEvent
{
    public:
        EndDebugEvent (Model& owner, bool showInTrace = false);
        virtual void    ExternalEventRoutine ();
};

// ---------------------------------------------------------------------------

class ReportEvent : public SystemEvent, public Observer
{
    public:
        ReportEvent (Model& owner, bool showInTrace = false);
        virtual void    ExternalEventRoutine ();
        virtual void    NoteChange (Observable*);
};

// ---------------------------------------------------------------------------

class ResetEvent : public SystemEvent, public Observer
{
    public:
        ResetEvent (Model& owner, bool showInTrace = false);
        virtual void    ExternalEventRoutine ();
        virtual void    NoteChange (Observable*);
};

// ---------------------------------------------------------------------------

#endif  // SYSTEMEVENT_H
```

sysevent.cc

```
// ---------------------------------------------------------------------------
//
// Datei
//          sysevent.cc
//
// Diplomarbeit
//
```

```
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#include "sysevent.h"

#include "experime.h"
#include "experimm.h"
#include "model.h"
#include "schedule.h"
#include "simclock.h"

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

SystemEvent::SystemEvent (Model& owner, const String& name, bool trace)
    :   ExternalEvent(owner, name, trace)
    {}

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

StopSimEvent::StopSimEvent (Model& owner, bool trace)
    :   SystemEvent (owner, "Stop Sim", trace)
    {}

// ---------------------------------------------------------------------------

void StopSimEvent::ExternalEventRoutine ()
    {
        //TraceNote("stops experiment");
        GetModel().GetExperiment().Stop (NOW());
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

StartTraceEvent::StartTraceEvent (Model& owner, bool trace)
    :   SystemEvent (owner, "Start Trace", trace)
    {}

// ---------------------------------------------------------------------------

void StartTraceEvent::ExternalEventRoutine ()
    {
        TraceOn();
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

EndTraceEvent::EndTraceEvent (Model& owner, bool trace)
    :   SystemEvent (owner, "Stop Trace", trace)
    {}

// ---------------------------------------------------------------------------

void EndTraceEvent::ExternalEventRoutine ()
    {
        TraceOff();
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

StartDebugEvent::StartDebugEvent (Model& owner, bool trace)
    :   SystemEvent (owner, "Start Debug", trace)
    {}

// ---------------------------------------------------------------------------

void StartDebugEvent::ExternalEventRoutine ()
    {
        DebugOn();
    }
```

```
// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

EndDebugEvent::EndDebugEvent (Model& owner, bool trace)
    :   SystemEvent (owner, "Stop Debug", trace)
    {}

// ---------------------------------------------------------------------------

void EndDebugEvent::ExternalEventRoutine ()
    {
        DebugOff();
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

ReportEvent::ReportEvent (Model& owner, bool trace)
    :   SystemEvent (owner, "Report", trace),
        Observer    ()
    {}

// ---------------------------------------------------------------------------

void ReportEvent::ExternalEventRoutine ()
    {
        // Nur bei der SimClock anmelden
        Observe (&ExperimentManager::Instance().GetSimClock (*this));
        // der Rest geschieht in 'NoteChange'
    }

// ---------------------------------------------------------------------------

void ReportEvent::NoteChange (Observable*)
    {
        // Report:
        ExperimentManager::Instance().Report (GetModel().GetExperiment());
        // abmelden:
        Observe();
        //Fuer die Zerstoerung vormerken:
        ExperimentManager::Instance().GetScheduler (*this).Terminate (*this);
    }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

ResetEvent::ResetEvent (Model& owner, bool trace)
    :   SystemEvent (owner, "Reset", trace),
        Observer    ()
    {}

// ---------------------------------------------------------------------------

void ResetEvent::ExternalEventRoutine ()
    {
        // Nur bei der SimClock anmelden
        Observe (&ExperimentManager::Instance().GetSimClock (*this));
        // der Rest geschieht in 'NoteChange'
    }

// ---------------------------------------------------------------------------

void ResetEvent::NoteChange (Observable*)
    {
        // Reset: Main-Model.Reset()
        GetModel().GetExperiment().GetModel().Reset();
        // abmelden:
        Observe();
        //Fuer die Zerstoerung vormerken:
        ExperimentManager::Instance().GetScheduler (*this).Terminate (*this);
    }

// ---------------------------------------------------------------------------
```

tally.h

```
// ---------------------------------------------------------------------------
//
```

```
// Datei
//          tally.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#ifndef TALLY_H
#define TALLY_H

// ---------------------------------------------------------------------------

#include "valuesta.h"   // Basisklasse

#include "valuesup.h"
#include "str.h"

// ---------------------------------------------------------------------------

class Tally : public ValueStatistics
{
    public:
                Tally (         Model&          owner,
                        const   String&         name,
                                ValueSupplier&  vs,
                                bool            showInReport = true,
                                bool            showInTrace  = false);
                Tally (         Model&          owner,
                                ValueSupplier&  vs,
                                bool            showInReport = true,
                                bool            showInTrace  = false);

        virtual void            Update ();
        virtual void            Reset ();
        virtual double          Mean () const;
        virtual double          StdDev () const;
        virtual Reporter*       NewReporter () const;

                String          ClassName () const;

    private:
        double  sum,
                sumSquare;
};

// ---------------------------------------------------------------------------

#endif  // TALLY_H
```

tally.cc

```
// ---------------------------------------------------------------------------
//
// Datei
//          tally.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
```

```
// ---------------------------------------------------------------------------

#include "tally.h"

#include <math.h>
#include "repstat.h"

// ---------------------------------------------------------------------------

static const char* className = "Tally";

// ---------------------------------------------------------------------------

Tally::Tally (          Model&          owner,
                        ValueSupplier&  vs,
                        bool            showInReport,
                        bool            showInTrace)
    :   ValueStatistics (owner, "", vs, showInReport, showInTrace),
        sum             (0),
        sumSquare       (0)
    {}

// ---------------------------------------------------------------------------

Tally::Tally (          Model&          owner,
                const   String&         name,
                        ValueSupplier&  vs,
                        bool            showInReport,
                        bool            showInTrace)

    :   ValueStatistics (owner, name, vs, showInReport, showInTrace),
        sum             (0),
        sumSquare       (0)
    {}

// ---------------------------------------------------------------------------

void Tally::Update ()
    {
        ValueStatistics::Update();

        double  value = Value();
        sum         += value;
        sumSquare   += value * value;
    }

// ---------------------------------------------------------------------------

void Tally::Reset ()
    {
        ValueStatistics::Reset();
        if (!Valid()) return;

        sum         =
        sumSquare   = 0;
    }

// ---------------------------------------------------------------------------

double Tally::Mean () const
    {
        const char* where = "Tally::Mean";

        if (!valid (className, where))
            return -1.0;

        if (Observations() == 0)
        {
            Warning (   "insufficient data",
                        where, "-1.0 is returned");
            return -1.0;
        }
        return sum / Observations();
    }

// ---------------------------------------------------------------------------

double Tally::StdDev () const
    {
        const char* where = "Tally::StdDev";

        if (!valid (className, where))
            return -1.0;
```

```
        long int n = Observations();
        if (n <= 1)
        {
            Warning (   "insufficient data",
                        where, "-1.0 is returned");
            return -1.0;
        }
        return sqrt (fabs (n * sumSquare - sum * sum)
                / (n * (n - 1)));
    }

// ---------------------------------------------------------------------------

Reporter* Tally::NewReporter () const
    {
        return new TallyReporter (*this);
    }

// ---------------------------------------------------------------------------

String Tally::ClassName () const
    {
        return className;
    }

// ---------------------------------------------------------------------------
```

text.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          text.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------
//
// Weiterentwicklung von:
//
// Diplomarbeit
//
//          Entwurf und Realisierung eines objektorientierten
//          Simulationspakets in C++
//
// Author
//          Heiko Weber
//
// Beschreibung
//
//          Die Klasse Text stellt vordefinierte Texte fuer SiFrane zur
//          Verfuegung. Hierdurch kann eine leichte Anpassung der
//          Ausgabesprache (Default = Englisch) ermoeglicht werden.
//
// ---------------------------------------------------------------------------

#ifndef  TEXT_H
#define  TEXT_H

// ---------------------------------------------------------------------------

#ifndef  TEXT_CC
#define  EXTERN(t, v, val)  extern t v
#else  //TEXT_CC
#define  EXTERN(t, v, val)  t v = val
#endif //TEXT_CC

// ---------------------------------------------------------------------------
```

```
EXTERN( const char *, txtSchedules,      "schedules");
EXTERN( const char *, txtOf,             "of");
EXTERN( const char *, txtNow,            "now");
EXTERN( const char *, txtAt,             "at");
EXTERN( const char *, txtItself,         "itself");
EXTERN( const char *, txtRemoves,        "removes");
EXTERN( const char *, txtInserts,        "inserts");
EXTERN( const char *, txtInto,           "into");
EXTERN( const char *, txtFrom,           "from");
EXTERN( const char *, txtFinds,          "finds");
EXTERN( const char *, txtReSchedules,    "reschedules");
EXTERN( const char *, txtCancels,        "cancels");
EXTERN( const char *, txtAfter,          "after");
EXTERN( const char *, txtBefore,         "before");
EXTERN( const char *, txtHoldsFor,       "holds for");
EXTERN( const char *, txtUntil,          "until");
EXTERN( const char *, txtTerminates,     "*** terminates");
EXTERN( const char *, txtPassivates,     "passivates");
EXTERN( const char *, txtAwaits,         "awaits");
EXTERN( const char *, txtSeizes,         "seizes");
EXTERN( const char *, txtGives,          "gives");
EXTERN( const char *, txtTo,             "to");
EXTERN( const char *, txtIn,             "in");
EXTERN( const char *, txtWaitsIn,        "waits in");
//EXTERN( const char *, txtCoOpts,         "co-opts");
EXTERN( const char *, txtNone,           "none");
EXTERN( const char *, txtSignals,        "signals");
EXTERN( const char *, txtLeaves,         "leaves");
EXTERN( const char *, txtBlockedIn,      "blocked in");
EXTERN( const char *, txtReleases,       "releases");
EXTERN( const char *, txtInterrupts,     "interrupts");
//EXTERN( const char *, txtWithReason,     "with reason");

// DESMO-C
EXTERN( const char *, txtNOW,            "NOW");
EXTERN( const char *, txtNullEntity,      "null-entity");
EXTERN( const char *, txtIt,              "it");
EXTERN( const char *, txtHasBeenIgnored,"has been ignored");
EXTERN( const char *, txtDeletes,        "deletes");
EXTERN( const char *, txtActivates,      "activates");
EXTERN( const char *, txtReActivates,    "reactivates");
EXTERN( const char *, txtInterruptSlave,"wants to interrupt slave");
EXTERN( const char *, txtInterruptWho,   "who...");
EXTERN( const char *, txtWithReason,     "cause:");
EXTERN( const char *, txtCoOpts,         "cooperates");
EXTERN( const char *, txtCoOptsFor,      "for");
EXTERN( const char *, txtWaitsFor,       "for");
EXTERN( const char *, txtTakes,          "takes");

// ------------------------------------------------------------------------------

#endif //TEXT_H
```

text.cc

```
// ------------------------------------------------------------------------------
//
// Datei
//         text.cc
//
// Diplomarbeit
//
//         DESMO-C
//         Implementierung eines Simulators fuer
//         zeitdiskrete Simulation in C++
//
// Autor
//         Thomas Schniewind
//
// Datum
//         8.3.1998
//
// ------------------------------------------------------------------------------
//
// uebernommen von:
//
// Diplomarbeit
```

```
//
//          Entwurf und Realisierung eines objektorientierten
//          Simulationspakets in C++
//
// Author
//          Heiko Weber
//
// Beschreibung
//
//          Die Klasse Text stellt vordefinierte Texte fuer SiFrane zur
//          Verfuegung. Hierdurch kann eine leichte Anpassung der
//          Ausgabesprache (Default = Englisch) ermoeglicht werden.
//
// ---------------------------------------------------------------------------

#define  TEXT_CC
#include "text.h"

// ---------------------------------------------------------------------------
```

timeseri.h

```
// ---------------------------------------------------------------------------
//
// Datei
//          timeseri.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#ifndef TIMESERIES_H
#define TIMESERIES_H

// ---------------------------------------------------------------------------

#include "statobj.h"    // Basisklasse

#include "valuesup.h"
#include "str.h"

#include <fstream.h>

// ---------------------------------------------------------------------------

class TimeSeries : public StatisticObject
{
        TimeSeries& operator= (const TimeSeries&);  // nicht implementiert
    public:
                    TimeSeries (Model&          owner,
                        const   String&         name,
                        const   String&         fileName,
                                ValueSupplier&  vs,
                        const   SimTime&        start       = 0.0,
                        const   SimTime&        end         = 0.0,
                                bool            automatic   = true,
                        const   String&         separator   = ",");
                        /*  'name' (falls nicht "") wird als erste Zeile in
                            die mit 'fileName' bezeichnete Datei geschrieben.
                            'fileName' muss den Namenskonvention des
                            unterliegenden Betriebssystems entprechen.
                            'start' und 'end' geben den Zeitraum an, in dem
                            der von 'vs' gelieferte Wert protokolliert wird.
                            'end' <= 'start' bedeuted, dass waehrend des
                            ganzen Experiments protokolliert wird.
                            Ist 'automatic' = true, so wird vor jedem Stellen
                            der Simulationsuhr protokolliert.
                            separator gibt an, wie Simulationszeit und Wert
```

```
                            getrennt werden sollen.
                        */
                    TimeSeries (const TimeSeries& objToCopy);

                double          Value () const;
        virtual void            Update ();
        virtual void            Reset ();
        virtual Reporter*       NewReporter () const;

                String          ClassName () const;

    private:
        String          fileName;
        ValueSupplier&  valueSupplier;
        SimTime         start, end;
        bool            automatic, allways;
        ofstream        os;
        String          separator;
};

// ----------------------------------------------------------------------------

#endif  // TIMESERIES_H
```

timeseri.cc

```
// ----------------------------------------------------------------------------
//
// Datei
//          timeseri.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ----------------------------------------------------------------------------

#include "timeseri.h"

#include "experimm.h"
#include "simclock.h"

// ----------------------------------------------------------------------------

static const char* className = "TimeSeries";

// ----------------------------------------------------------------------------

TimeSeries::TimeSeries (Model&          owner,
                const   String&         name,
                const   String&         FileName,
                        ValueSupplier&  vs,
                const   SimTime&        Start,
                const   SimTime&        End,
                        bool            Automatic,
                const   String&         Separator)
    :   StatisticObject(owner, name, false, false),
        fileName        (FileName),
        valueSupplier   (vs),
        start           (Start),
        end             (End),
        automatic       (Automatic),
        allways         (End <= Start),
        os              (FileName.Get()),
        separator       (Separator)
    {
        const char* where = "";

        if (automatic)
            Observe (&ExperimentManager::Instance().GetSimClock (*this));
```

```
        if (!os)
            //  Warnung: Datei konnte nicht geoeffnet werden
            Warning (   "file '" + fileName + "' could not be opened",
                        where);
        else
            if (Name() != "")
                os << Name() << endl;
    }

// ---------------------------------------------------------------------------

TimeSeries::TimeSeries (const TimeSeries& ts)
    :   StatisticObject (ts),
        fileName        (ts.fileName),
        valueSupplier   (ts.valueSupplier),
        start           (ts.start),
        end             (ts.end),
        automatic       (ts.automatic),
        allways         (ts.allways),
        os              (ts.fileName.Get(), ios::out || ios::app),
        separator       (ts.separator)
    {
        const char* where = "";

        if (!os)
            Warning (   "file '" + fileName + "' could not be opened",
                        where);
        else
            if (Name() != "")
                os << Name() << endl;
    }

// ---------------------------------------------------------------------------

double TimeSeries::Value () const
    {
        const char* where = "TimeSeries::Value";

        if (!valid (className, where))
            return -1.0;
        if (!valid (valueSupplier, "ValueSupplier", where))
            return -1.0;

        return valueSupplier.Value();
    }

// ---------------------------------------------------------------------------

void TimeSeries::Update ()
    {
        const char* where = "TimeSeries::Update";

        if (!valid (className, where))
            return;

        SimTime t = CurrentTime();

        if (!allways)
        {
            if (t < start)  return;
            if (t > end)
            {
                if (automatic)
                    Observe (0);
                return;
            }
        }

        if (!valid (valueSupplier, "ValueSupplier", where))
            return;

        IncObservations();
        if (os)
            os << t << separator << valueSupplier.Value() << endl;
        traceUpdate();
    }

// ---------------------------------------------------------------------------

void TimeSeries::Reset ()
    {
        StatisticObject::Reset();
        if (!Valid()) return;
```

```
        if (os)
        {
            os.close();
            os.open (fileName.Get());
            if (Name() != "" && os)
                os << Name() << endl;
        }
    }

// -----------------------------------------------------------------------------

Reporter* TimeSeries::NewReporter () const
    {
        return 0;
    }

// -----------------------------------------------------------------------------

String TimeSeries::ClassName () const
    {
        return className;
    }

// -----------------------------------------------------------------------------
```

valuesta.h

```
// -----------------------------------------------------------------------------
//
// Datei
//          valuesta.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// -----------------------------------------------------------------------------

#ifndef VALUESTATISTICS_H
#define VALUESTATISTICS_H

// -----------------------------------------------------------------------------

#include "statobj.h"    // Basisklasse

#include "valuesup.h"
#include "str.h"

// -----------------------------------------------------------------------------

class ValueSupplier;

// -----------------------------------------------------------------------------

class ValueStatistics : public StatisticObject
{
    public:
                    ValueStatistics(        Model& owner,
                                    const   String& name,
                                            ValueSupplier& vs,
                                    bool    showInReport = true,
                                    bool    showInTrace  = false);
                    ValueStatistics(        Model& owner,
                                            ValueSupplier& vs,
                                    bool    showInReport = true,
                                    bool    showInTrace  = false);

        virtual void        Update ();

                double      Value ()    const;
```

```
               double      Minimum ()  const;
               double      Maximum ()  const;
        virtual double      Mean ()     const = 0;
        virtual double      StdDev ()   const = 0;

        virtual void        Reset ();

                String      ClassName () const;

    private:
        ValueSupplier&      valueSupplier;
        double              min,
                            max,
                            lastValue;
};

// ---------------------------------------------------------------------------

#endif  // VALUESTATISTICS_H
```

valuesta.cc

```
// ---------------------------------------------------------------------------
//
// Datei
//          valuesta.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#include "valuesta.h"

#include "valuesup.h"

// ---------------------------------------------------------------------------

static const char* className = "ValueStatistics";

// ---------------------------------------------------------------------------

ValueStatistics::ValueStatistics (  Model&          owner,
                            const   String&         name,
                                    ValueSupplier&  vs,
                                    bool            showInReport,
                                    bool            showInTrace)
    :   StatisticObject (owner, name, showInReport, showInTrace),
        valueSupplier   (vs),
        min             (0),
        max             (0),
        lastValue       (0)
    {
    }

// ---------------------------------------------------------------------------

ValueStatistics::ValueStatistics (  Model&          owner,
                                    ValueSupplier&  vs,
                                    bool            showInReport,
                                    bool            showInTrace)
    :   StatisticObject(owner, "", showInReport, showInTrace),
        valueSupplier(vs),
        min(0),
        max(0),
        lastValue(0)
    {
    }
```

```
// --------------------------------------------------------------------------

double ValueStatistics::Value () const
    {
        return lastValue;
    }

// --------------------------------------------------------------------------

double ValueStatistics::Minimum () const
    {
        return min;
    }

// --------------------------------------------------------------------------

double ValueStatistics::Maximum () const
    {
        return max;
    }

// --------------------------------------------------------------------------

void ValueStatistics::Reset ()
    {
        StatisticObject::Reset();
        if (!Valid()) return;

        min         =
        max         =
        lastValue   = 0;
    }

// --------------------------------------------------------------------------

String ValueStatistics::ClassName () const
    {
        return className;
    }

// --------------------------------------------------------------------------

void ValueStatistics::Update ()
    {
        const char* where = "ValueStatistics::Update";

        if (!valid (className, where))
            return;
        if (!valid (valueSupplier, "ValueSupplier", where))
            return;

        lastValue = valueSupplier.Value();
        IncObservations();
        if (Observations() <= 1)
            min = max = lastValue;
        else if (lastValue < min)
            min = lastValue;
        else if (lastValue > max)
            max = lastValue;
        traceUpdate();
    }

// --------------------------------------------------------------------------
```

valuesup.h

```
// --------------------------------------------------------------------------
//
// Datei
//          valuesup.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
```

```
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#ifndef VALUESUPPLIER_H
#define VALUESUPPLIER_H

// ---------------------------------------------------------------------------

#include "modelcom.h"   // Basisklasse
#include "observab.h"   // Basisklasse

#include "str.h"

// ---------------------------------------------------------------------------

class ValueSupplier : public ModelComponent
{
    public:
                    ValueSupplier (Model& owner, const String& name = "");
        virtual    ~ValueSupplier ();

        virtual double      Value () const  = 0;

                String      ClassName () const;
};

// ---------------------------------------------------------------------------

#endif  // VALUESUPPLIER_H
```

valuesup.cc

```
// ---------------------------------------------------------------------------
//
// Datei
//          valuesup.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#include "valuesup.h"

// ---------------------------------------------------------------------------

static const char* className = "ValueSupplier";

// ---------------------------------------------------------------------------

ValueSupplier::ValueSupplier (Model& owner, const String& name)
    :   ModelComponent(owner, name)
    {}

// ---------------------------------------------------------------------------

ValueSupplier::~ValueSupplier ()
    {}

// ---------------------------------------------------------------------------

String ValueSupplier::ClassName () const
    {
        return className;
    }
```

```
// --------------------------------------------------------------------------
```

waitq.h

```
// --------------------------------------------------------------------------
//
// Datei
//          waitq.h
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// --------------------------------------------------------------------------

#ifndef WAITQUEUE_H
#define WAITQUEUE_H

// --------------------------------------------------------------------------

#include "qbased.h" // Basisklasse

#include "boolean.h"
#include "conditio.h"
#include "coop.h"
#include "process.h"
#include "pqueue.h"
#include "simtime.h"
#include "str.h"

// --------------------------------------------------------------------------

class QueueImpl;

// --------------------------------------------------------------------------

class WaitQueue : public QueueBased
    /*  WaitQueues dienen der Synchronisation fuer die Prozesskooperation.
        Master teilen ihren Kooperationswusch mittels 'Cooperate', Slaves
        mittels 'Wait' mit. Die gemeinsamen Handlungen werden dem Master
        zugeordnet. Ist kein passender Prozess vorhanden, wird der Prozess
        in eine implizite Warteschlange (je eine fuer Master und Slaves)
        eingereiht.
    */
{
        friend  class WaitQueueReporter;

        WaitQueue&  operator= (const WaitQueue&);
                        // Zuweisung nicht implementiert
    public:
                    WaitQueue ( Model&  owner,
                        const   String& name         = "",
                                bool    showInReport = true,
                                bool    showInTrace  = true);
                    WaitQueue (const WaitQueue&);
        virtual    ~WaitQueue ();

                void            Wait();
                void            Cooperate (ProcessCooperation& coop);
                void            Cooperate (ProcessCooperation& coop,
                                           Condition&          cond);
                                           // DESMO: Find (...)
                bool            Avail     (Process*& slave, Condition& c);

                // Informationen ueber die implizite Master-Warteschlange:
                // wird ueber die Oberklasse QueueBased abgewickelt
                bool            mEmpty()            const;

                unsigned long   mLength()           const;
```

```
                unsigned long   mMinLength()        const;
                unsigned long   mMaxLength()        const;
                double          mAvgLength()        const;
                double          mStdDevLength()     const;
                SimTime         mMinLengthAt()      const;
                SimTime         mMaxLengthAt()      const;

                SimTime         mZeroWaits()        const;
                SimTime         mMaxWaitTime()      const;
                SimTime         mAvgWaitTime()      const;
                SimTime         mStdDevWaitTime()   const;
                SimTime         mMaxWaitTimeAt()    const;

                // Informationen ueber die implizite Slave-Warteschlange:
                bool            sEmpty()            const;

                unsigned long   sLength()           const;
                unsigned long   sMinLength()        const;
                unsigned long   sMaxLength()        const;
                double          sAvgLength()        const;
                double          sStdDevLength()     const;
                SimTime         sMinLengthAt()      const;
                SimTime         sMaxLengthAt()      const;

                SimTime         sZeroWaits()        const;
                SimTime         sMaxWaitTime()      const;
                SimTime         sAvgWaitTime()      const;
                SimTime         sStdDevWaitTime()   const;
                SimTime         sMaxWaitTimeAt()    const;

                Reporter*       NewReporter() const;

                String          ClassName () const;
    protected:

    private:
                bool        checkProcess   (Process&    p,
                                            const char* where) const;
                bool        checkCondition (const Condition&,
                                            const char* where) const;
                void        activateAsNext (Process&    p,
                                            const char* where) const;
                void        activateFirst  (const char* where) const;

        QueueImpl&      qimpl;          // wartende Master
        ProcessQueue    waitingSlaves;  // wartende Slaves
};

// ---------------------------------------------------------------------------

#endif  // WAITQUEUE_H
```

waitq.cc

```
// ---------------------------------------------------------------------------
//
// Datei
//          waitq.cc
//
// Diplomarbeit
//
//          DESMO-C
//          Implementierung eines Simulators fuer
//          zeitdiskrete Simulation in C++
//
// Autor
//          Thomas Schniewind
//
// Datum
//          8.3.1998
//
// ---------------------------------------------------------------------------

#include "waitq.h"

#include "conditio.h"
#include "expopts.h"
#include "pblocker.h"
```

```
#include "process.h"
#include "qimpl.h"
#include "repwaitq.h"
#include "msgwaitq.h"

#include <assert.h>

// ---------------------------------------------------------------------------

static const char* className = "WaitQueue";

// ---------------------------------------------------------------------------

WaitQueue::WaitQueue (  Model&  owner,
                const   String& name,
                        bool    showInReport,
                        bool    showInTrace)
    :   QueueBased      (owner, "", showInReport, showInTrace),
        qimpl           (*new QueueImpl (*this)),
        waitingSlaves   (owner, "", false, false) // weder trc noch rep
    {
        unsigned n = GetExperimentOpts().NameWidth() - 2;
                     Rename (name.Left (n) + " M");
        waitingSlaves.Rename (name.Left (n) + " S");
    }

// ---------------------------------------------------------------------------

WaitQueue::WaitQueue (const WaitQueue& wq)
    :   QueueBased      (wq),
        qimpl           (*new QueueImpl (*this)),
        waitingSlaves   (wq.waitingSlaves)
    {}

// ---------------------------------------------------------------------------

WaitQueue::~WaitQueue ()
    {
        delete &qimpl;
    }

// ---------------------------------------------------------------------------

bool WaitQueue::checkProcess (Process& p, const char* where) const
    {
        if (!p.Valid ())
        {
            Warning ("invalid object", where);
            return false;
        }
        if (p.IsNullProcess())
        {
            Warning ("only processes may use a WaitQueue",
                     where);
            return false;
        }
        if (!IsExperimentCompatible (p))
        {
            Warning ("attemp to mix components of different experiments",
                     where, "ignored");
            return false;
        }
        if (!IsModelCompatible (p))
        {
            Warning ("incompatible process", where);
            return false;
        }
        return true;
    }

// ---------------------------------------------------------------------------

bool WaitQueue::checkCondition (const Condition& c, const char* where) const
    {
        if (!valid (c, "Condition", where))
            return false;
        if (!IsExperimentCompatible (c))
        {
            Warning ("attemp to mix components of different experiments",
                     where, "ignored");
            return false;
        }
        if (!IsModelCompatible (c))
        {
```

```
            Warning (   "incompatible " + c.ClassName(), where, "ignored");
            return false;
        }
        return true;
    }

// --------------------------------------------------------------------------

void WaitQueue::activateAsNext (Process& process, const char* where) const
    {
        if (!process.IsNullProcess())
        {
            if (!checkProcess (process, where))
                return;
            if (process.IsScheduled())
            {   // anders als in DESMO!
                process.SkipTraceNote ();
                process.Cancel();
            }

            bool wasBlocked = process.Blocked();
            if (wasBlocked)
                ProcessBlocker::UnBlock (process);
                    // um Aktivierung zu erlauben

            process.SkipTraceNote ();
            process.ActivateAfter (Current());

            if (wasBlocked) ProcessBlocker::SetBlocked (process);
        }
    }

// --------------------------------------------------------------------------

void WaitQueue::activateFirst (const char* where) const
    {
        Process& process = (Process&)qimpl.First (where);

        if (!process.IsNullProcess())
        {
            if (!checkProcess (process, where))
                return;
            if (process.IsScheduled())
            {   // anders als in DESMO!
                process.SkipTraceNote ();
                process.Cancel();
            }

            bool wasBlocked = process.Blocked();
            if (wasBlocked)
                ProcessBlocker::UnBlock (process);
                    // um Aktivierung zu erlauben

            process.SkipTraceNote ();
            process.ActivateAfter (Current());

            if (wasBlocked) ProcessBlocker::SetBlocked (process);
        }
    }

// --------------------------------------------------------------------------

void WaitQueue::Wait()
    {
        const char* where = "WaitQueue::Wait";

        if (!valid (className, where))
            return;

        Process& slave = CurrentProcess();
        if (!checkProcess (slave, where))
            return;

        if (slave.waitQueue)
        {
            Warning (   "slave already waits in WaitQueue "
                        + slave.waitQueue->QuotedName(),
                        where);
            return;
        }

        if (TraceIsOn())
            SendMessage (TrcWaitQWait (waitingSlaves));
```

```
        waitingSlaves.Insert (slave);

        if (Length() > 0)
            activateFirst (where);

        slave.waitQueue = &waitingSlaves;   // wird in Process::Cooperate
        ProcessBlocker::Block (slave);      // zurueckgesetzt
    }

// ------------------------------------------------------------------------------

void WaitQueue::Cooperate (ProcessCooperation& coop)
    {
        const char* where = "WaitQueue::Cooperate";

        if (!valid (className, where))
            return;

        Process& master = CurrentProcess();
        if (!checkProcess (master, where))
            return;

        qimpl.Insert (master, where);

        if (    waitingSlaves.Length() == 0
            ||  &master != &qimpl.First (where))
        {
            if (TraceIsOn())
                SendMessage (TrcWaitQWait (*this));

            if (waitingSlaves.Length() > 0)
                activateAsNext ((Process&)qimpl.First (where), where);

            do
            {   // blockieren, solange keine Slaves warten
                ProcessBlocker::Block (master);
            } while (waitingSlaves.Length() <= 0);
        }
        activateAsNext ((Process&)qimpl.Succ (master, where), where);

        qimpl.Remove (master, where);
        ProcessBlocker::UnBlock (master);

        Process& slave = waitingSlaves.First();
        if (!checkProcess (slave, where))
            return;
        slave.Cooperate (coop);
    }

// ------------------------------------------------------------------------------

void WaitQueue::Cooperate (ProcessCooperation& coop, Condition& cond)
    {
        const char* where = "WaitQueue::Cooperate";

        if (!valid (className, where))
            return;

        Process& master = CurrentProcess();
        Process* slave  = &NullProcess();
        if (!checkProcess (master, where))
            return;
        if (!checkCondition (cond, where))
            return;

        qimpl.Insert (master, where);

        if (    &master != &qimpl.First (where)
            ||  !Avail (slave, cond))
        {
            if (TraceIsOn())
                SendMessage (TrcWaitQWaitFor (*this, cond));

            if (    waitingSlaves.Length() > 0
                &&  &master != &qimpl.First (where))
                activateAsNext ((Process&)qimpl.First (where), where);

            do
            {
                ProcessBlocker::Block (master);
                if (Avail (slave, cond))
                    break;
                if (waitingSlaves.Length() > 0)
                    activateAsNext ((Process&)qimpl.Succ (master, where),
```

```
                                   where);
            } while (true);
        }
        // Kooperationsbedingung erfuellt
        assert (slave);
        if (waitingSlaves.Length() > 1)
            activateAsNext ((Process&)qimpl.Succ (master, where), where);

        qimpl.Remove (master, where);
        ProcessBlocker::UnBlock (master);

        if (!checkProcess (*slave, where))
            return;
        if (TraceIsOn())
        {
            SendMessage (TrcWaitQFind (*slave, waitingSlaves, coop, cond));
            SkipTraceNote();
        }
        slave->Cooperate (coop);
    }

// ---------------------------------------------------------------------------

bool WaitQueue::Avail (Process*& slave, Condition& cond)
    {
        const char* where = "WaitQueue::Avail";

        if (!valid (className, where))
            return false;

        Process& master = CurrentProcess();
        if (!checkProcess (master, where))
            return false;
        if (!checkCondition (cond, where))
            return false;

        slave = &waitingSlaves.First (cond);
        return !slave->IsNullProcess();
    }

// ---------------------------------------------------------------------------
// Informationen ueber die implizite Master-Warteschlange:

bool WaitQueue::mEmpty() const
    { return Empty(); }

// ---------------------------------------------------------------------------

unsigned long WaitQueue::mLength() const
    { return Length(); }

// ---------------------------------------------------------------------------

unsigned long WaitQueue::mMinLength() const
    { return MinLength(); }

// ---------------------------------------------------------------------------

unsigned long WaitQueue::mMaxLength() const
    { return MaxLength(); }

// ---------------------------------------------------------------------------

double WaitQueue::mAvgLength() const
    { return AvgLength(); }

// ---------------------------------------------------------------------------

double WaitQueue::mStdDevLength() const
    { return StdDevLength(); }

// ---------------------------------------------------------------------------

SimTime WaitQueue::mMinLengthAt() const
    { return MinLengthAt(); }

// ---------------------------------------------------------------------------

SimTime WaitQueue::mMaxLengthAt() const
    { return MaxLengthAt(); }

// ---------------------------------------------------------------------------

SimTime WaitQueue::mZeroWaits() const
```

```
    { return ZeroWaits(); }

// ---------------------------------------------------------------------------

SimTime WaitQueue::mMaxWaitTime() const
    { return MaxWaitTime(); }

// ---------------------------------------------------------------------------

SimTime WaitQueue::mAvgWaitTime() const
    { return AvgWaitTime(); }

// ---------------------------------------------------------------------------

SimTime WaitQueue::mStdDevWaitTime() const
    { return StdDevWaitTime(); }

// ---------------------------------------------------------------------------

SimTime WaitQueue::mMaxWaitTimeAt() const
    { return MaxWaitTimeAt(); }

// ---------------------------------------------------------------------------
// Informationen ueber die implizite Slave-Warteschlange:

bool WaitQueue::sEmpty() const
    { return waitingSlaves.Empty(); }

// ---------------------------------------------------------------------------

unsigned long WaitQueue::sLength() const
    { return waitingSlaves.Length(); }

// ---------------------------------------------------------------------------

unsigned long WaitQueue::sMinLength() const
    { return waitingSlaves.MinLength(); }

// ---------------------------------------------------------------------------

unsigned long WaitQueue::sMaxLength() const
    { return waitingSlaves.MaxLength(); }

// ---------------------------------------------------------------------------

double WaitQueue::sAvgLength() const
    { return waitingSlaves.AvgLength(); }

// ---------------------------------------------------------------------------

double WaitQueue::sStdDevLength() const
    { return waitingSlaves.StdDevLength(); }

// ---------------------------------------------------------------------------

SimTime WaitQueue::sMinLengthAt() const
    { return waitingSlaves.MinLengthAt(); }

// ---------------------------------------------------------------------------

SimTime WaitQueue::sMaxLengthAt() const
    { return waitingSlaves.MaxLengthAt(); }

// ---------------------------------------------------------------------------

SimTime WaitQueue::sZeroWaits() const
    { return waitingSlaves.ZeroWaits(); }

// ---------------------------------------------------------------------------

SimTime WaitQueue::sMaxWaitTime() const
    { return waitingSlaves.MaxWaitTime(); }

// ---------------------------------------------------------------------------

SimTime WaitQueue::sAvgWaitTime() const
    { return waitingSlaves.AvgWaitTime(); }

// ---------------------------------------------------------------------------

SimTime WaitQueue::sStdDevWaitTime() const
    { return waitingSlaves.StdDevWaitTime(); }

// ---------------------------------------------------------------------------
```

```
SimTime WaitQueue::sMaxWaitTimeAt() const
    { return waitingSlaves.MaxWaitTimeAt(); }

// ---------------------------------------------------------------------------

Reporter* WaitQueue::NewReporter() const
    {
        return new WaitQueueReporter (*this);
    }

// ---------------------------------------------------------------------------

String WaitQueue::ClassName () const
    {
        return className;
    }

// ---------------------------------------------------------------------------
```

Verzeichnis der Klassennamen

T

V

W